Links between Probabilistic Automata and Hidden Markov Models: probability distributions, learning models and induction algorithms

P. Dupont[a,*] F. Denis[b] Y. Esposito[b]

[a]*INGI, Univesity of Louvain,*
Place Sainte-Barbe 2,
B-1348 Louvain-la-Neuve, Belgium

[b]*LIF-CMI, UMR 6166,*
39, Rue F. Joliot Curie,
13453 Marseille Cedex, France

Abstract

This article[a] presents an overview of Probabilistic Automata (PA) and discrete Hidden Markov Models (HMMs), and aims at clarifying the links between them. The first part of this work concentrates on probability distributions generated by these models. Necessary and sufficient conditions for an automaton to define a probabilistic language are detailed. It is proved that probabilistic deterministic automata (PDFA) form a proper subclass of probabilistic non-deterministic automata (PNFA). Two families of equivalent models are described next. On one hand, HMMs and PNFA with no final probabilities generate distributions over complete finite prefix-free sets. On the other hand, HMMs with final probabilities and probabilistic automata generate distributions over strings of finite length. The second part of this article presents several learning models, which formalize the problem of PA induction or, equivalently, the problem of HMM topology induction and parameter estimation. These learning models include the PAC and identification with probability 1 frameworks. Links with Bayesian learning are also discussed. The last part of this article presents an overview of induction algorithms for PA or HMMs using state merging, state splitting, parameter pruning and error-correcting techniques.

[a] A DRAFT was published as *P. Dupont, F. Denis and Y. Esposito, Links between Probabilistic Automata and Hidden Markov Models: probability distributions, learning models and induction algorithms, UCL-INGI Research Report RR2003-02, January 2003.*

Key words: Probabilistic Automata, Hidden Markov Models, Grammar Induction, PAC learning, Bayesian learning, Induction algorithms, HMM topology learning

[*] Corresponding author. Tel: +32 210 479114; Fax: +32 210 450345.
Email addresses: pdupont@info.ucl.ac.be (P. Dupont), fdenis@cmi.univ-mrs.fr (F. Denis), esposito@cmi.univ-mrs.fr (Y. Esposito).
URLs: www.info.ucl.ac.be/~pdupont/ (P. Dupont), www.cmi.univ-mrs.fr/~fdenis/ (F. Denis).

1 Introduction

Hidden Markov Models (HMMs) are widely used in many pattern recognition areas, including applications to speech recognition [4,42,53,36], biological sequence modeling [24,8], information extraction [59] and optical character recognition [43], to name a few. In many of these cases, the model structure, also referred to as topology, is defined according to some prior knowledge of the application domain. In some cases however, attempts are made to induce automatically the model structure from training data. The learning problem combines then structural induction and parameter estimation.

Grammar Induction, also known as Grammatical Inference, is a collection of techniques for learning grammars from training data [28,46,69,58]. Early works on grammar induction already covered learning techniques for probabilistic (or stochastic[1]) grammars [34,29,45,20]. Probabilistic regular grammars form a particular class of interest. These models are equivalent to certain types of probabilistic automata (PA), for which several induction techniques have been proposed [57,13,55,56,14,64].

This article presents an overview of probabilistic automata and discrete Hidden Markov Models, and aims at clarifying the links between them. These links allow to apply induction techniques and learnability results developed in one formalism to the other.

The first part of this work (sections 2 and 3) concentrates on probability distributions generated by PA and HMMs. Necessary and sufficient conditions for an automaton to define a probabilistic language are detailed. The distinction between probabilistic deterministic automata (PDFA) and probabilistic non-deterministic automata (PNFA) is introduced. This distinction matters for the learning problem as it is proved in section 3 that PDFA form a proper subclass of PNFA. Two families of equivalent models are described next. On one hand, HMMs and PNFA with no final probabilities generate distributions over complete finite prefix-free sets. On the other hand, HMMs with final probabilities and probabilistic automata generate distributions over strings of finite length.

The second part of this article (sections 4 and 5) presents several learning models. Learning a probabilistic automaton aims, in a broad sense, at inducing an automaton generating a distribution $\hat{P}$ from a sample drawn according to some unknown target distribution P. The distribution $\hat{P}$ forms the learned hypothesis that approximates the target. The purpose of a learning model is to formalize the notion of learning when a specific quality measure defines the distance between P and $\hat{P}$. We discuss adaptations of the PAC learning and identification in the limit frameworks to the learning of probabilistic automata. Links with Bayesian learning are also discussed. A learning model includes a learning protocol specifying the prior knowledge given to the learner, the required quality of the proposed hypothesis, and, possibly, some bounds on the computational complexity of the learning process. Once a learning model has been defined, the question of what can be learned by any algorithm following the learning protocol, can be addressed. Several learning results are presented in this context in section 5.

The last part of this article (section 6) presents an overview of induction algorithms for PA or HMMs. State merging is a generalization technique starting from an initial model fitting

[1] We consider that the term *stochastic* qualifies a process, while the term *probabilistic* qualifies a model of such process. We use therefore the term probabilistic grammars (or probabilistic automata) since we consider them as models.

perfectly a given learning sample. An opposite approach is state splitting where a very general model is progressively specialized to best fit the training data. Structural induction can also be embedded into parameter estimation combined with parameter pruning. Finally, error-correcting techniques greedily adapt an initial structure by minimizing some edition costs to best incorporate new samples.

2 Probabilistic languages, automata and HMMs

Probabilistic languages are defined in section 2.1. We discuss in section 2.2 various equivalent definitions of semi-probabilistic automata. The main result of section 2.3 is the proposition 2 which establishes the necessary and sufficient conditions for a semi-probabilistic automaton to be probabilistic, that is, to define a distribution on words (or strings). Probabilistic automata considered in the present work can be considered as a representation of probabilistic regular grammars (see *e.g.* [20]). The notions of probabilistic non-deterministic versus deterministic automata are introduced next. This distinction matters, as demonstrated in section 3, for the class of distributions generated by the later form a proper subclass of the class of distributions generated by the former. Section 2.4 concentrates on probabilistic automata with no final probabilities and details the type of distributions they generate. Hidden Markov Models are described in section 2.5.

2.1 Probabilistic languages

Notations

Σ denotes a finite *alphabet*, Σ^* (respectively Σ^∞) denotes the set of words of finite (respectively infinite) length over Σ. For any word $u \in \Sigma^*$, $u\Sigma^*$ (respectively $u\Sigma^\infty$) denotes the set of finite (respectively infinite) words with prefix u. ε denotes the *empty word* and $|u|$ the *length* of a word u. For any $n \in \mathbb{N}$, Σ^n (respectively $\Sigma^{\leq n}$) denotes the set of words of length n (respectively less or equal to n).

Definition 1 *Let Σ be a finite alphabet, a* semi-distribution *over Σ^* is a function $\psi : \Sigma^* \rightarrow [0,1]$ satisfying $\sum_{u\in\Sigma^*} \psi(u) \leq 1$.*

Definition 2 *The* support *$L_\psi \subseteq \Sigma^*$ of the semi-distribution ψ is the language $L_\psi = \{u \in \Sigma^* | \psi(u) > 0\}$.*

Definition 3 *A* distribution *or* probabilistic language *ψ over Σ^* is a semi-distribution such that $\sum_{u\in\Sigma^*} \psi(u) = 1$.*

2.2 Semi-probabilistic automata

Definition 4
A semi-probabilistic automaton [2] *(semi-PA) is a 5-tuple $\langle \Sigma, Q, \phi, \iota, \tau \rangle$ where Σ is a finite al-*

[2] Such an automaton is called a semi-PA and not a PA as it defines a semi-distribution (see Corollary 1). The supplementary conditions to be satisfied to define a distribution are detailed in definition 9.

phabet, Q is a finite set of states, $\phi : Q \times \Sigma \times Q \to [0,1]$ is a mapping defining the transition probability function, $\iota : Q \to [0,1]$ is a mapping defining the initial probability of each state, and $\tau : Q \to [0,1]$ is a mapping defining the final probability of each state. The following constraints must be satisfied:

$$\sum_{q\in Q} \iota(q) = 1 \textit{ and } \forall q \in Q, \tau(q) + \sum_{a\in\Sigma}\sum_{q'\in Q} \phi(q,a,q') = 1.$$

A state q is said to be initial *if $\iota(q) > 0$ and* final *if $\tau(q) > 0$.*

A_q denotes the automaton $\langle \Sigma, Q, \phi, \iota_q, \tau\rangle$ where $q \in Q$ and $\iota_q(q') = 1$ if $q = q'$, and 0 otherwise.

Definition 5
The symbol ϕ also denotes two extensions of the transition function, respectively defined on $Q \times \Sigma^* \times Q$:

$$\phi(q,\varepsilon,q') = \begin{cases} 1 \textit{ if } q = q' \\ 0 \textit{ otherwise} \end{cases}$$

$$\forall u \in \Sigma^*, \forall a \in \Sigma, \phi(q,ua,q') = \sum_{q''\in Q} \phi(q,u,q'')\phi(q'',a,q')$$

and on $Q \times 2^{\Sigma^*} \times 2^Q$:

$$\phi(q,U,Q') = \sum_{u\in U}\sum_{q'\in Q'} \phi(q,u,q').$$

$\phi(q,u,q')$ can be interpreted as the probability of reaching state q' from state q while generating the word u.

Definition 6 *Let $A = \langle \Sigma, Q, \phi, \iota, \tau\rangle$ be a semi-PA.*
The functions $P_A : \Sigma^* \to [0,1]$ and $\overline{P}_A : \Sigma^* \to [0,1]$ are defined as follows:

$$P_A(u) = \sum_{q,q'\in Q} \iota(q)\phi(q,u,q')\tau(q')$$

and

$$\overline{P}_A(u) = \sum_{q,q'\in Q} \iota(q)\phi(q,u,q').$$

$P_A(u)$ can be interpreted as the probability of generating word u. $\overline{P}_A(u)$ can be interpreted as the probability of generating a (possibly infinite) word with prefix u. For all word u, $\overline{P}_{A_q}(u) = \phi(q,u,Q)$. The functions P_A and $\overline{P}_A$ can be extended to subsets U of Σ^*:

$$P_A(U) = \sum_{u\in U} P_A(u) \text{ and } \overline{P}_A(U) = \sum_{u\in U} \overline{P}_A(u),\ \forall U \subseteq \Sigma^*. \tag{1}$$

For any word u, the following equality is satisfied:

$$\overline{P}_A(u) = P_A(u) + \overline{P}_A(u\Sigma). \tag{2}$$

Lemma 1 *Let A be a semi-PA. For any integer n, we have*

$$P_A(\Sigma^{\leq n}) + \overline{P}_A(\Sigma^{n+1}) = 1.$$

PROOF. According to equation (2), for any integer k we have

$$\overline{P}_A(\Sigma^k) = P_A(\Sigma^k) + \overline{P}_A(\Sigma^{k+1}).$$

Lemma 1 follows from adding up the preceding equalities for k varying between 0 and n, and from noting that $\overline{P}_A(\varepsilon) = 1$. □

Corollary 1 *Let A be a semi-PA, $P_A : \Sigma^* \to [0,1]$ defines a semi-distribution over Σ^*.*

PROOF. According to lemma 1, $\overline{P}_A(\Sigma^n)$ is a decreasing series for increasing values of n. It follows that

$$P_A(\Sigma^*) = 1 - \lim_{n\to\infty} \overline{P}_A(\Sigma^n) \leq 1.$$

□

Definition 7 *Two semi-probabilistic automata are* equivalent *if they define the same semi-distribution.*

Proposition 1 *Any semi-PA is equivalent to a semi-PA with a single initial state.*

PROOF. Let $A = \langle \Sigma, Q, \phi, \iota, \tau \rangle$ be a semi-PA. $A' = \langle \Sigma, Q', \phi', \iota', \tau' \rangle$ is defined as follows:

$$Q' = Q \cup \{q_0\} \text{ where } q_0 \text{ is a new state}$$

$$\forall a \in \Sigma, \phi'(q,a,q') = \begin{cases} \phi(q,a,q') & \text{if } q, q' \in Q \\ 0 & \text{if } q' = q_0 \\ \sum_{q'' \in Q} \iota(q'')\phi(q'',a,q') & \text{if } q = q_0, q' \in Q. \end{cases}$$

$$\iota'(q) = \begin{cases} 1 & \text{if } q = q_0 \\ 0 & \text{otherwise.} \end{cases}$$

$$\tau'(q) = \begin{cases} \sum_{q' \in Q} \iota(q')\tau(q') & \text{if } q = q_0 \\ \tau(q) & \text{otherwise.} \end{cases}$$

It follows that

$$\begin{aligned} \tau'(q_0) + \sum_{a\in\Sigma}\sum_{q'\in Q'} \phi'(q_0,a,q') &= \sum_{q\in Q} \iota(q)\tau(q) + \sum_{a\in\Sigma}\sum_{q'\in Q}\sum_{q\in Q} \iota(q)\phi(q,a,q') \\ &= \sum_{q\in Q} \iota(q)\left[\tau(q) + \sum_{a\in\Sigma}\sum_{q'\in Q} \phi(q,a,q')\right] = \sum_{q\in Q} \iota(q) = 1. \end{aligned}$$

One can easily check that A' is a semi-PA. Moreover we have:

$$P_A(\varepsilon) = \sum_{q\in Q} \iota(q)\tau(q) = \iota'(q_0)\tau'(q_0) = P_{A'}(\varepsilon),$$

and, for any word u and any letter a, we have:

$$\begin{aligned}
P_A(au) &= \sum_{q,q' \in Q} \iota(q)\phi(q, au, q')\tau(q') \\
&= \sum_{q,q',q'' \in Q} \iota(q)\phi(q, a, q'')\phi(q'', u, q')\tau(q') \\
&= \sum_{q',q'' \in Q} (\sum_{q \in Q} \iota(q)\phi(q, a, q''))\phi(q'', u, q')\tau(q') \\
&= \sum_{q',q'' \in Q} \phi'(q_0, a, q'')\phi(q'', u, q')\tau(q') \\
&= \sum_{q',q'' \in Q} \iota'(q_0)\phi'(q_0, a, q'')\phi'(q'', u, q')\tau'(q') \\
&= P_{A'}(au).
\end{aligned}$$

A and A' define therefore the same semi-distribution. □

The construction above is illustrated by the examples presented in figures 1 and 2. Given the constraints on τ and ϕ, the function τ is redundant as $\forall q \in Q, \tau(q) = 1 - \sum_{a\in\Sigma} \sum_{q'\in Q} \phi(q, a, q')$. Thus a semi-PA can be equivalently defined as a 4-tuple $A = \langle\Sigma, Q, \phi, q_0\rangle$ with the constraint $\sum_{a\in\Sigma} \sum_{q'\in Q} \phi(q, a, q') \leq 1$. Following a similar construction, it can be shown that any semi-PA is equivalent to a semi-PA with a single initial state and a single final state, provided one considers a special *end-of-word* symbol for reaching the final state (see for example [56]). Since all these definitions are equivalent, we use in the sequel definition 4.

2.3 Probabilistic automata

In this section we characterize which semi-probabilistic automata are defining distributions on words.

Definition 8 *A state q of a semi-PA A is* accessible *if $\phi(Q_I, \Sigma^*, q) > 0$, where Q_I is the set of initial states of A. Otherwise, q is* unaccessible.

The set of accessible states can be obtained in linear time. The semi-distribution associated to a semi-PA remains unchanged if all unaccessible states are removed.

A probabilistic automaton A is a semi-PA such that the probability of reaching a final state from any accessible state is strictly positive.

Definition 9 *A semi-PA A is a* probabilistic automaton *(PA) if for any accessible state q,*

$$P_{A_q}(\Sigma^*) = \sum_{q'} \phi(q, \Sigma^*, q')\tau(q') > 0.$$

Definition 10 *A PA is* trimmed *if all of its states are accessible.*

Given any PA, an equivalent trimmed PA may be constructed in linear time.

Lemma 2 *Let $A = \langle\Sigma, Q, \phi, \iota, \tau\rangle$ be a PA with n states. If state q is accessible then*

$$\phi(q, \Sigma^n, Q) < 1.$$

PROOF. By definition of a PA having n states, there exists a final state q' accessible from q by a word u of length $\leq n-1$. In other words, $P_{A_q}(\Sigma^{<n}) > 0$. It follows that

$$\phi(q, \Sigma^n, Q) = \overline{P}_{A_q}(\Sigma^n) = 1 - P_{A_q}(\Sigma^{<n}) < 1.$$

□

Proposition 2 *Let A be a semi-PA, A is a PA if and only if P_A is a distribution.*

PROOF. Let A be a PA with n states. Without loss of generality, we can assume A to be trimmed. Let α be defined as $\alpha = \max\{\phi(q, \Sigma^n, Q) | q \in Q\}$. According to lemma 2, $\alpha < 1$. We show by recurrence on k that, for any state q, $\phi(q, \Sigma^{kn}, Q) \leq \alpha^k$.

$$\begin{aligned}\phi(q, \Sigma^{kn}, Q) &= \sum_{q' \in Q} \phi(q, \Sigma^n, q')\phi(q', \Sigma^{(k-1)n}, Q) \\ &\leq \alpha^{k-1} \sum_{q' \in Q} \phi(q, \Sigma^n, q') \\ &= \alpha^{k-1}\phi(q, \Sigma^n, Q) \\ &\leq \alpha^k.\end{aligned}$$

It follows that

$$\lim_{k\to\infty} \overline{P}_A(\Sigma^{kn}) \leq \lim_{k\to\infty} \alpha^k = 0$$

Hence, according to corollary 1, P_A is a distribution.

Let A be a semi-PA such that P_A is a distribution. Q_I denotes the set of initial states of A. Let q be an accessible state of A, and let v, with $|v| = l$, be a word such that $\phi(Q_I, v, q) > 0$. For any $n \in \mathbb{N}$, we have

$$\begin{aligned}\overline{P}_A(\Sigma^{n+l}) &\geq \overline{P}_A(v\Sigma^n) \\ &\geq \phi(Q_I, v, q)\overline{P}_{A_q}(\Sigma^n) \\ &\geq \phi(Q_I, v, q)(1 - P_{A_q}(\Sigma^{<n})) \geq 0.\end{aligned}$$

As $\overline{P}_A(\Sigma^{n+l})$ tends to 0 when n tends to infinity, $P_{A_q}(\Sigma^{<n})$ tends to 1. Thus according to definition 9, A is a PA. □

Definition 11 *The* support automaton *of a PA $A = \langle \Sigma, Q, \phi, \iota, \tau \rangle$ is a non-deterministic finite automaton (NFA) $\underline{A} = \langle \Sigma, Q, \delta, I, F \rangle$ where I (respectively F) denotes the set of initial (respectively final) states of A, and $\delta \subseteq Q \times \Sigma \times Q$ denotes the transition function defined as follows: $(q, a, q') \in \delta \Leftrightarrow \phi(q, a, q') > 0$*

A direct consequence of this definition is that the language L generated by the support automaton of a PA A is the support of the distribution P_A. In the sequel, we call PNFA (respectively PDFA) a PA the support of which is a non-deterministic finite automaton (NFA) (respectively a deterministic finite automaton (DFA)).

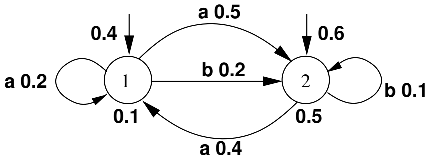

Figure 1. A PNFA example.

Figure 1 presents a PNFA defined as follows:

- $\Sigma = \{a, b\}$
- $Q = \{1, 2\}$
- $\phi(1, a, 1) = 0.2; \phi(1, b, 1) = 0; \phi(1, a, 2) = 0.5; \phi(1, b, 2) = 0.2;$ $\phi(2, a, 1) = 0.4; \phi(2, b, 1) = 0; \phi(2, a, 2) = 0; \phi(2, b, 2) = 0.1$
- $\iota(1) = 0.4; \iota(2) = 0.6$
- $\tau(1) = 0.1; \tau(2) = 0.5$

For instance the probability of word b is given by:

$$\begin{aligned} P_A(b) &= \iota(1)\phi(1, b, 1)\tau(1) \, + \, \iota(1)\phi(1, b, 2)\tau(2) \\ &+ \, \iota(2)\phi(2, b, 1)\tau(1) \, + \, \iota(2)\phi(2, b, 2)\tau(2) \\ &= 0.07 \end{aligned}$$

Here the support language is $(a + b)^*$.

Figure 2 presents an equivalent PNFA with a single initial state.

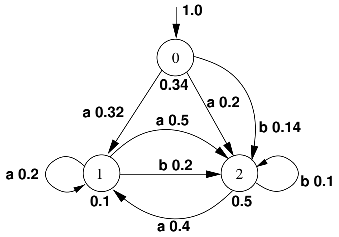

Figure 2. A PNFA with a single initial state.

Definition 12 *A probabilistic language is* regular *if it can be generated by a PNFA The class $\mathcal{P}NFA$ denotes the class of probabilistic regular languages.*

As the support of a probabilistic regular language (PRL) must be a regular language, it is clear that there exist probabilistic languages that are not regular [3]. There exist also probabilistic languages, with regular support languages, that are not PRL [4].

Definition 13 *A probabilistic regular language is* deterministic *if it can be generated by a PDFA. The class $\mathcal{P}DFA$ denotes the class of probabilistic deterministic regular languages (PDRL).*

$\mathcal{P}DFA$ is a proper subclass of $\mathcal{P}NFA$ (see proposition 5), which is an important result for the learning of probabilistic automata. Another interesting subclass of PDRL are the probabilistic

[3] Consider for instance the class of probabilistic context-free languages [35,71].

[4] Consider for instance the regular support language $L = \{a^*\}$, and the distribution $\psi(a^n) = \frac{1}{e.n!}, \forall n \geq 0$.

finite support languages.

Proposition 3 *Every probabilistic language having a finite support is in $\mathcal{PDFA}$.*

PROOF. Let ψ be a probabilistic language over Σ with a finite support. We define the automaton $A = \langle \Sigma, Q, \phi, \iota, \tau \rangle$ where Q is the set of prefixes of words in the support of ψ, the unique initial state is ε, and for any words u and v of Q and any letter a, $\tau(u) = \psi(u)/\psi(u\Sigma^*)$ and

$$\phi(u, a, v) = \begin{cases} \psi(v\Sigma^*)/\psi(u\Sigma^*) \text{ if } v = ua \\ 0 \text{ otherwise.} \end{cases}$$

It is easy to show that A generates the language ψ. □

The PDFA defined in proposition 3 is the probabilistic prefix tree acceptor used for state merging induction techniques (see section 6.2.1).

Probabilistic automata used in the present work can be seen as probabilistic generators. They are equivalent to probabilistic regular grammars [27,29,31]. These automata differ from probabilistic acceptors (see for example [50,15]) and are not equivalent [31]. In the case of a probabilistic acceptor (or recognizer), there is an input alphabet Σ and an output alphabet Y. A probabilistic acceptor[5] defines a *conditional* probability $P(Y = y|u)$, for a given word u of Σ^*.

We focus on probabilistic generators, which define *unconditional* distributions over Σ^*, and study their links with HMMs (see section 3). Some interesting links between HMMs and probabilistic acceptors are described in [7], but the notion of distribution equivalence is distinct from ours. In particular an HMM M and a probabilistic acceptor A are considered equivalent if the probabilities of generating words by M are the same as accepting them by A. In our case we ask for equal generation probabilities.

2.4 Probabilistic automata with no final probabilities

A particular type of probabilistic automata do not include final probabilities (see for instance [2,39]). They can be defined in our formalism as follows.

Definition 14 *A* probabilistic automaton with no final probabilities *(NFPA) is a semi-PA where the set of final states is empty.*

Let $A = \langle \Sigma, Q, \phi, \iota, \tau \rangle$ be a NFPA. According to definition 14, we have: $\forall q \in Q, \tau(q) = 0$. Thus, for any word u, $P_A(u) = 0$ and $\overline{P}_A(u)$ can be interpreted as the probability of generating an infinite word starting with the prefix u. A NFPA defines therefore a probability on the (continuous) space of infinite words Σ^∞.

According to lemma 1, we have $\overline{P}_A(\Sigma^n) = 1$ and a NFPA defines one distribution for each value of n. More generally, we obtain a probabilistic language for any restriction of $\overline{P}_A$ to a complete

[5] The output alphabet is usually binary $Y = \{0, 1\}$, and the value $Y = 1$ (respectively $Y = 0$) is then associated to final (respectively non-final) states. In this case, the string u is said to be *accepted* if $P(Y = 1|u) > 0$.

finite prefix-free set.

Definition 15 *A set of words $U \subseteq \Sigma^*$ is* prefix-free *if no word of U is a prefix of another word in U. More formally, we have*

$$\forall u, v \in U, \exists w \in \Sigma^*, v = uw \Rightarrow w = \varepsilon.$$

A prefix-free set U is complete *if all word $u \in \Sigma^*$ has a prefix in U or is a prefix of a word in U.*

A complete prefix-free set is maximal with respect to inclusion among the family of prefix-free sets. For example, the set $\{a^n b \mid n \in \mathbb{N}\}$ is complete prefix-free if $\Sigma = \{a, b\}$, as it is the case for each set $\{\Sigma^n\}$, for any value of n.

Proposition 4 *If A is a NFPA and U is a prefix-free set then $\overline{P}_A$ defines a semi-distribution on U. If U is moreover complete finite then $\overline{P}_A$ defines a distribution on U.*

PROOF. Let $A = \langle \Sigma, Q_A, \phi, \iota, \tau \rangle$ be a NFPA and let U be a prefix-free set.

For any word $u \in \Sigma^*$, we have $\overline{P}_A(u) = \sum_{a \in \Sigma} \overline{P}_A(ua)$. This implies that $\overline{P}_A(u) = \overline{P}_A(u\Sigma^k)$ for any integer k.

For any integer n, we have

$$\begin{aligned}\overline{P}_A(U \cap \Sigma^{\leq n}) &= \sum_{v \in U \cap \Sigma^{\leq n}} \overline{P}_A(v\Sigma^{n-|v|}) \\ &\leq \overline{P}_A(\Sigma^n) = 1.\end{aligned}$$

In the limit when n tends to ∞, we obtain $\overline{P}_A(U) \leq 1$.

Moreover if U is finite, we can consider $n \geq \max\{|u|, u \in U\}$. It follows that

$$\overline{P}_A(U) = \overline{P}_A(U \cap \Sigma^{\leq n}) = \sum_{v \in U} \overline{P}_A(v\Sigma^{n-|v|})$$

Now, if U is complete, for any word u in Σ^n, there exists necessarily a word of U that is a prefix of u. We obtain

$$\overline{P}_A(U) = \overline{P}_A(\Sigma^n) = 1.$$

□

Note that the previous proposition does not hold if U is infinite. Consider for instance a NFPA A such that $\overline{P}_A(a^n) = 1$ for any integer n and the set $U = \{a^n b \mid n \in \mathbb{N}\}$.

2.5 Hidden Markov Models

Definition 16 *A discrete* Hidden Markov Model (HMM) *(with state emission) is a 5-tuple $M = \langle \Sigma, Q, A, B, \iota \rangle$ where Σ is an alphabet, Q is a set of states, $A : Q \times Q \to [0, 1]$ is a mapping defining the probability of each transition, $B : Q \times \Sigma \to [0, 1]$ is a mapping defining the emission* *probability of each letter on each state, and $\iota : Q \rightarrow [0,1]$ is a mapping defining the initial probability of each state. The following constraints must be satisfied:*

$$\forall q \in Q, \sum_{q' \in Q} A(q,q') = 1$$
$$\forall q \in Q, \sum_{a \in \Sigma} B(q,a) = 1$$
$$\sum_{q \in Q} \iota(q) = 1$$

Definition 17 *Let $M = \langle \Sigma, Q, A, B, \iota \rangle$ be a HMM. A* path *in M is a word defined on Q^*. For any path ν, ν_i denotes the i-th state of ν, and $|\nu|$ denotes the path length. For any word $u \in \Sigma^*$ and any path $\nu \in Q^*$, the probabilities $P_M(u,\nu)$ and $P_M(u)$ are defined as follows:*

$$P_M(u,\nu) = \begin{cases} \iota(\nu_1) \prod_{i=1}^{l-1} [B(\nu_i, u_i) A(\nu_i, \nu_{i+1})] B(\nu_l, u_l) \text{ if } l = |u| = |\nu| > 0, \\ 1 \text{ if } |u| = |\nu| = 0 \text{ and} \\ 0 \text{ otherwise.} \end{cases}$$
$$P_M(u) = \sum_{\nu \in Q^*} P(u,\nu).$$

$P_M(u,\nu)$ is the probability to emit word u while following path ν. Along any path, the emission process is markovian since the probability of emitting a letter on a given state only depends on that state. HMMs are used to model processes for which the existence of such a path (or state sequence) can be assumed while the actual states are not observed. $P_M(u)$ can be interpreted as the probability of observing a finite prefix u of some infinite word.

Alternative definitions of HMMs (see for example [15,24]) include a single non-emitting initial state q_0, also called a *silent state*, and transitions of q_0 to the other states, as well as a non-emitting final state q_f and transitions from the other states to q_f. The use of a single initial state q_0, with initial probability $\iota(q_0) = 1$, results in models equivalent to HMMs described here (the proof is analogous to the one used to demonstrate proposition 1). On the other hand, the introduction of a non-emitting final state modifies the associated distributions. Proposition 9 in section 3 explains this result.

Figure 3 presents a HMM defined as follows.

- $\Sigma = \{a, b\}$
- $Q = \{1, 2\}$
- $A(1,1) = 0.1; A(1,2) = 0.9; A(2,1) = 0.7; A(2,2) = 0.3$
- $B(1,a) = 0.2; B(1,b) = 0.8; B(2,a) = 0.9; B(2,b) = 0.1$
- $\iota(1) = 0.4; \iota(2) = 0.6$

For instance, the probability of the word ab is given by:

$$\begin{aligned} P_M(ab) &= P_M(ab, 11) + P_M(ab, 12) + P_M(ab, 21) + P_M(ab, 22) \\ &= 0.0064 + 0.0072 + 0.3024 + 0.0162 \\ &= 0.3322. \end{aligned}$$

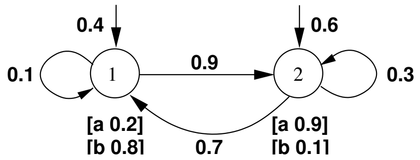

Figure 3. An example of HMM (with emission on states).

Hidden Markov Models can also be defined with emissions on transitions [5,15] instead of states.

Definition 18 *A discrete* Hidden Markov Model with transition emission (HMMT) *is a 5-tuple $M = \langle \Sigma, Q, A, B, \iota \rangle$, where Σ is an alphabet, Q is a set of states, $A : Q \times Q \to [0,1]$ is a mapping defining the probability of each transition, $B : Q \times \Sigma \times Q \to [0,1]$ is a mapping defining the emission probability of each letter on each transition, and $\iota : Q \to [0,1]$ is a mapping defining the initial probability of each state. The following constraints must be satisfied:*

$$\forall q \in Q, \sum_{q' \in Q} A(q,q') = 1$$

$$\forall q, q' \in Q, \sum_{a \in \Sigma} B(q,a,q') = \begin{cases} 1 \text{ if } A(q,q') > 0 \\ 0 \text{ otherwise.} \end{cases}$$

$$\sum_{q \in Q} \iota(q) = 1.$$

Definition 19 *Let $M = \langle \Sigma, Q, A, B, \iota \rangle$ be a HMMT. A* path *in M is a word defined on Q^*. For any word $u \in \Sigma^*$ and any path $\nu \in Q^*$, the probabilities $P_M(u,\nu)$ and $P_M(u)$ are defined as follows:*

$$P_M(u,\nu) = \begin{cases} \iota(\nu_1) \prod_{i=1}^{|u|} [B(\nu_i, u_i, \nu_{i+1}) A(\nu_i, \nu_{i+1})] \; \textit{if } |\nu| = |u|+1, \; \textit{and} \\ 0 \; \textit{otherwise,} \end{cases}$$

$$P_M(u) = \sum_{\nu \in Q^*} P(u,\nu).$$

Figure 4 presents a HMMT defined as follows.

- $\Sigma = \{a, b\}$
- $Q = \{1, 2\}$
- $A(1,1) = 0.1; A(1,2) = 0.9; A(2,1) = 0.7; A(2,2) = 0.3$
- $B(1,a,1) = 0.2; B(1,b,1) = 0.8; B(1,a,2) = 0.3; B(1,b,2) = 0.7; B(2,a,1) = 0.8; B(2,b,1) = 0.2; B(2,a,2) = 0.9; B(2,b,2) = 0.1$
- $\iota(1) = 0.4; \iota(2) = 0.6$

For instance, the probability of the word b is given by:

$$\begin{aligned} P_M(b) &= \iota(1)B(1,b,1)A(1,1) + \iota(1)B(1,b,2)A(1,2) \\ &+ \iota(2)B(2,b,1)A(2,1) + \iota(2)B(2,b,2)A(2,2) \\ &= 0.386 \end{aligned}$$

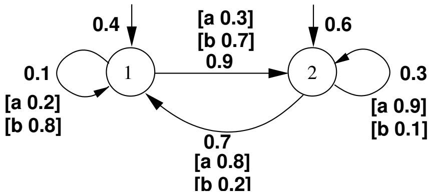

Figure 4. An example of HMMT (with emission on transitions).

Definitions 16 and 18 are similar to the definitions of probabilistic automata. We clarify the links between these models in section 3. Note that we consider here HMMs defined on a discrete alphabet. Many variants can be found in the literature, including models with a continuous emission density, typically defined by a Gaussian or a multi-Gaussian instead of a discrete (multinomial) distribution (see, for example, [51], [52], [53], [36]).

3 Links between PDFA, PNFA and HMMs

We study in this section the relations between the distributions generated by PDFA, PNFA and HMMs. Proposition 5 shows that the class of probabilistic deterministic regular languages, which are generated by PDFA, forms a proper subclass of the class of probabilistic regular languages, which are generated by PNFA. Propositions 8 and 9 show the equivalence between PNFA, HMMTs and HMMs. The constructive proofs given here illustrate how to transform any such model into any of both others.

Proposition 5 $\mathcal{PDFA} \subsetneq \mathcal{PNFA}$.

PROOF.

Let A be a probabilistic automaton and let define $\rho(u)$ as follows:

$$\forall u \in \Sigma^*, \rho(u) = \begin{cases} \frac{P_A(u)}{\overline{P}_A(u)} & \text{, if } \overline{P}_A(u) > 0 \\ 0 & \text{, otherwise.} \end{cases}$$

If A is a PDFA, the set $\{\rho(u), u \in \Sigma^*\}$ is necessarily finite.

Consider now the PNFA described in Figure 5. We have $\rho(a^n) = 0.6 + \frac{0.2}{1+2^n}$, which is a strictly decreasing series for strictly increasing values of n. Hence $\{\rho(u), u \in \Sigma^*\}$ cannot be finite.

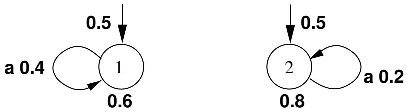

Figure 5. A PNFA generating a language that cannot be generated by a PDFA.

□

The proof of proposition 5 uses an ambiguous[6] PNFA which cannot be reduced to a PDFA. Proposition 6 shows that the same result hold even if one considers the class of non-ambiguous PNFA ($na\mathcal{P}NFA$) and proposition 7 shows that this class is a proper subclass of $\mathcal{P}NFA$. Hence, proposition 5 is thus also directly implied by propositions 6 and 7.

Proposition 6 *$\mathcal{P}DFA \subsetneq na\mathcal{P}NFA$.*

PROOF. Consider the non-ambiguous PNFA described in Figure 6. In this case, $\rho(a^{2n}) = 0.6 - \frac{0.6}{1+2^n}$ which is a strictly decreasing series for strictly increasing values of n.

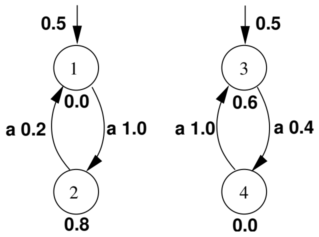

Figure 6. A non-ambiguous PNFA generating a language that cannot be generated by a PDFA.

□

Proposition 7 *$na\mathcal{P}NFA \subsetneq \mathcal{P}NFA$.*

PROOF. Let ψ be the probabilistic language defined on $\Sigma = \{a\}$ by

$$\psi(a^n) = \frac{0.6(0.4)^n + 0.8(0.2)^n}{2}.$$

This language is generated by the ambiguous PNFA described in Figure 5. Suppose that there exists a non-ambiguous PNFA $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ such that $\psi = P_A$ and let s be the number of states of A. Let $q_0, \ldots, q_s$ be the unique state sequence generating a^s and let $i < j$ be two indexes such that $q_i = q_j$. Let

$$\alpha = \iota(q_0) \left[\prod_{k=0}^{i-1} \varphi(q_k, a, q_{k+1}) \right] \left[\prod_{k=j}^{s-1} \varphi(q_k, a, q_{k+1}) \right] \tau(q_s)$$

and let $\beta = \prod_{k=i}^{j-1} \varphi(q_k, a, q_{k+1})$. Since A is non-ambiguous, we must have $\psi(a^{s+m(j-i)}) = \alpha\beta^m$ for all integer m which is clearly impossible. □

Next we show the equivalence between probabilistic automata with no final probabilities and Hidden Markov Models.

Lemma 3 *Let $A = \langle \Sigma, Q, \phi, \iota, \tau \rangle$ be a PNFA with no final probabilities. There exists an equivalent HMMT $M = \langle \Sigma, Q, A, B, \iota \rangle$.*

[6] A PNFA is ambiguous if there exists at least one word that can be generated by several state sequences.

PROOF. Σ, Q and ι are identical for A and M. The transition functions for M are defined as follows.

- $\forall q, q' \in Q, A(q, q') = \sum_{a \in \Sigma} \phi(q, a, q')$
- $\forall q, q' \in Q, \forall a \in \Sigma, B(q, a, q') = \begin{cases} \frac{\phi(q,a,q')}{\sum_{a \in \Sigma} \phi(q,a,q')} & \text{if } \sum_{a \in \Sigma} \phi(q, a, q') > 0 \\ 0 & \text{otherwise.} \end{cases}$

It is easily shown that M satisfies the constraints of a HMMT and that M and A generate the same distribution.

□

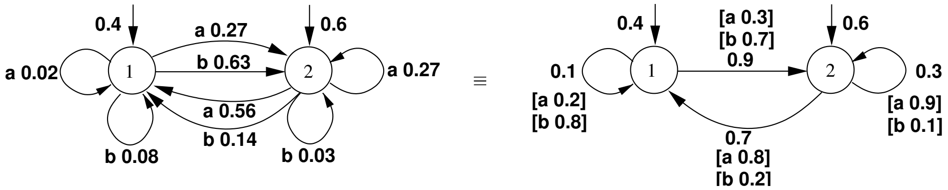

Figure 7. Transformation of a PNFA into an equivalent HMMT.

Lemma 4 *Let* $M = \langle \Sigma, Q, A, B, \iota \rangle$ *be a HMMT, there exists an equivalent HMM* $M' = \langle \Sigma, Q', A', B', \iota' \rangle$.

PROOF.

The construction of a HMM equivalent to a HMMT is given in [15]. In this case, the number of states $|Q'|$ is less or equal to $|Q|^2$. M' is defined as follows.

- $Q' = \{(q, q') \in Q \times Q | A(q, q') > 0\}$. The states of Q' represents pairs of states in Q that are connected by a strictly positive transition probability.
- $\forall (q, q'), (q'', q''') \in Q', A((q, q'), (q'', q''')) = \begin{cases} A(q'', q''') & \text{if } q' = q'' \\ 0 & \text{otherwise.} \end{cases}$
- $\forall (q, q') \in Q', \forall a \in \Sigma, B((q, q'), a) = B(q, a, q')$
- $\forall (q, q') \in Q', \iota'((q, q')) = \iota(q) A(q, q')$

It is easily shown that M' satisfies the constraints of a HMM and that M and M' generate the same distribution.

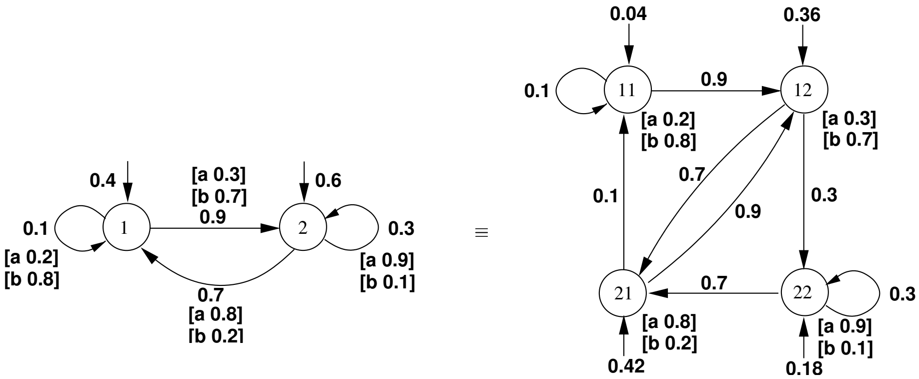

Figure 8. Transformation of a HMMT into an equivalent HMM (first construction).

We give below an alternative construction for which the number of states $|Q|'$ is less or equal to $|Q| \times |\Sigma|$. Let M' be defined as follows.

- $Q' = Q \times \Sigma$,
- $\iota'((q,a)) = \sum_{q' \in Q} \iota(q')A(q',q)B(q',a,q)$,
- $B'((q,a),x) = 1$ if $x = a$, and 0 otherwise,
- $A'((q,a),(q',b)) = A(q,q')B(q,b,q')$.

It is easily shown that M' satisfies the constraints of a HMM.

Let $u = u_1 \dots u_l$ be a word of Σ^* and let $\nu = ((q_1,u_1)\dots(q_l,u_l))$ be a path in M'. We have:

$$\begin{aligned} P_{M'}(u,\nu) &= \iota'((q_1,u_1)) \prod_{i=1}^{l-1} [B'((q_i,u_i),u_i)A'((q_i,u_i),(q_{i+1},u_{i+1}))]B'((q_l,u_l),u_l) \\ &= \sum_{q' \in Q} \iota(q')A(q',q_1)B(q',u_1,q_1) \prod_{i=1}^{l-1} [A((q_i,q_{i+1})B(q_i,u_{i+1},q_{i+1})] \\ &= \sum_{q' \in Q} P_M(u, q'q_1 \dots q_l). \end{aligned}$$

Summing up over all possible paths in M', we obtain $P_{M'}(u) = P_M(u)$. Hence, M and M' generate the same distribution. □

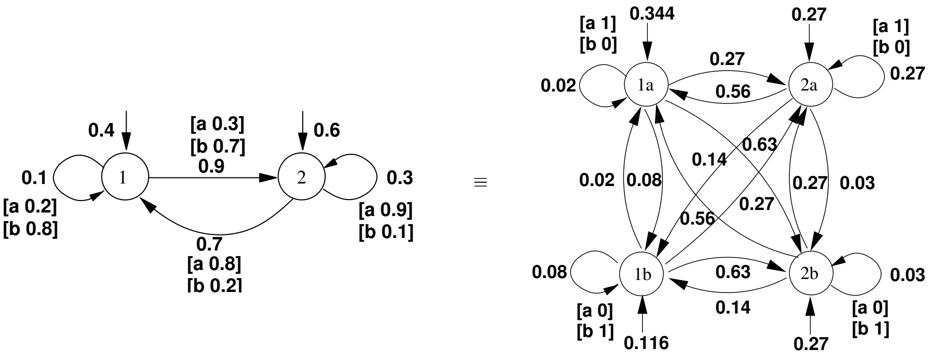

Figure 9. Transformation of a HMMT into an equivalent HMM (second construction).

The number of degrees of freedom (parameters) of a HMM (respectively a HMMT) with n states over an alphabet of m letters is $n-1+n(m-1)+n(n-1) = n^2+nm-n-1 \in \mathcal{O}(n \times \max(n,m))$ (respectively $n-1+n(n-1)+n^2(m-1) = n^2m-1 \in \mathcal{O}(n^2m)$). Hence the transformation of a HMMT into an equivalent HMM cannot be performed in general without changing the number of states.

Lemma 5 *Let $M = \langle \Sigma, Q, A, B, \iota \rangle$ be a HMM. There exists an equivalent PNFA with no final probabilities $A = \langle \Sigma, Q, \phi, \iota, \tau \rangle$.*

PROOF. A is defined as follows.

- $\forall q, q' \in Q, \forall a \in \Sigma, \phi(q,a,q') = B(q,a)A(q,q')$
- $\forall q \in Q, \tau(q) = 0$

It is easily shown that A satisfies the constraints of a PNFA and that A and M generates the same distribution. □

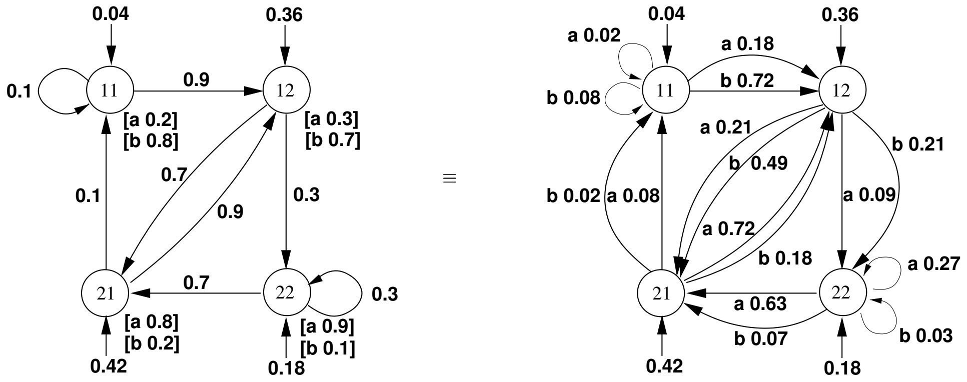

Figure 10. Transformation of a HMM into an equivalent PNFA.

Proposition 8 *Hidden Markov Models are equivalent to probabilistic automata with no final probabilities.*

PROOF. This is a direct consequence of lemmas 3, 4 and 5. □

The equivalence between these models is demonstrated using constructive proofs to transform a PNFA into a HMMT (lemma 3), a HMMT into a HMM (lemma 4), and a HMM into a PNFA (lemma 5). Note that the PNFA of figure 10 is not isomorphic to the PNFA of figure 7, even though they generate the same distribution. The possibility to simulate a HMM with n states by a PNFA with n states was already proved in [2]. Proposition 8 guarantees that one can simulate a PNFA by a HMM but not, in general, with the same number of states. However the sizes of all these equivalent models are always polynomially related.

Corollary 2 *If M is a HMM or a HMMT, then $\forall n \in \mathbb{N}$, $\sum_{u \in \Sigma^n} P_M(u) = 1$.*

PROOF.

This result, mentioned in [2], is a direct consequence of propositions 4 and 8. □

Definitions 16 and 18 correspond to Hidden Markov Models with no final probabilities. Variants of these models, including a final non-emitting state q_f, correspond to models where a final probability $\tau(q)$ is defined for each state. Hence the proposition 9 follows.

Proposition 9 *Hidden Markov Models with final probabilities are equivalent to semi-probabilistic automata.*

PROOF. The demonstration of this result is completely analogous to the proof of proposition 8. □

Corollary 3 *Hidden Markov Models with final probabilities, and such that the probability of reaching a final state from any accessible state is strictly positive, generate distributions over Σ^*.*

PROOF. This is a direct consequence of propositions 2 and 9. □

To sum up, there are two families of equivalent models. On one hand, HMMs and probabilistic automata with no final probabilities, which generate distributions over $\Sigma^n, \forall n \in \mathbb{N}$, or, more generally, over any complete finite prefix-free set. On the other hand, HMMs with final probabilities and probabilistic automata, which generate distributions over Σ^*.

4 Learning models

Learning a probabilistic automaton aims, in a broad sense, at inducing an automaton generating a distribution $\hat{P}$ from a sample drawn according to some unknown target distribution P. The distribution $\hat{P}$ forms the learned hypothesis that approximates the target. The purpose of a *learning model* is to formalize the notion of learning when a specific quality measure defines the distance between P and $\hat{P}$.

A learning model includes a *learning protocol* specifying the prior knowledge given to the learner, the required quality of the proposed hypothesis, and, possibly, some bounds on the computational complexity of the learning process. Given a learning model, the question of what can be learned by any algorithm following the learning protocol, can be addressed.

In the context of probabilistic automaton learning, an important particular case occurs when the prior knowledge includes the support automaton of the target distribution (see definition 11). Prior knowledge, generally coming from the application domain, enables to fix *a priori* the structure of the target automaton or an equivalent HMM topology. In this case, the set of free parameters is fixed and learning is then reduced to the problem of estimating probabilities given a known structure. The more general case, studied as well in the sequel, occurs when probability estimation is combined with structural induction.

Several models for learning probabilistic automata are presented in this section. Learning results obtained in these models are presented in section 5.

4.1 A PAC learning model for probabilistic automata

The PAC[7] learning model was introduced by Valiant [67]. We focus here on various adaptations of this model when the concepts to be learned are probabilistic automata [39,55,56].

Definition 20 *Let P be a target distribution and let $\hat{P}$ be a hypothesis produced by a learning algorithm. Let D be a measure of the distance*[8] *between P and $\hat{P}$.*

[7] PAC learning stands for *Probably Approximately Correct* learning.

[8] This distance is not necessarily a metric.

$\hat{P}$ *is an* ϵ-good hypothesis *with respect to* P*, for* $\epsilon \geq 0$*, if*

$$D(P, \hat{P}) \leq \epsilon$$

Kearns et al. use the Kullback-Leibler divergence D_{KL} as distance measure between P and $\hat{P}$:

$$D_{KL}(P, \hat{P}) = \sum_u P(u) \log_2 \frac{P(u)}{\hat{P}(u)} \tag{3}$$

where the summation is over all words belonging to the domain of P, assumed to be identical to the domain of $\hat{P}$. The divergence can be interpreted as the number of additional bits needed to encode a message when an optimal code is chosen according to distribution $\hat{P}$ while the message was produced according to distribution P. This measure bounds the L_1 distance[9] and the Hellinger distance D_H [10].

Let $\mathcal{P}(.)$ denote a distribution class. Each distribution P of the class $\mathcal{P}(.)$ represents a concept, the size of which, denoted by $|P|$, depends polynomially on a set of parameters. For example, $\mathcal{P}DFA_{|\Sigma|,|Q|}$ is the class of distributions that can be generated by PDFA defined on an alphabet of size $|\Sigma|$ and having $|Q|$ states. $|\Sigma|$ and $|Q|$ are the parameters characterizing the size of each concept in the class. These automata form the *representation class* of the corresponding distributions.

Definition 21 *A distribution class* $\mathcal{P}(.)$ *is* efficiently learnable *if there exists a learning algorithm satisfying the following conditions. For any target distribution* $P \in \mathcal{P}(.)$*, the algorithm receives an independent and identically distributed (iid) sample* S_P *from* P*, a precision parameter* $\epsilon > 0$ *and a confidence parameter* δ*,* $0 < \delta \leq 1$*. The algorithm outputs, with probability at least* $1 - \delta$*, an* ϵ*-good hypothesis* $\hat{P}$ *with respect to* P*. The time complexity of the learning algorithm has to be a polynomial function of* $\frac{1}{\epsilon}, \frac{1}{\delta}$ *and* $|P|$*.*

Definition 21 specifies the learning protocol of a distribution learning algorithm. It should be remarked that the representation classes of the target distribution P and the hypothesis $\hat{P}$ need not be the same. The representation class issue is further studied below.

4.2 A trainability model for probabilistic automata

Abe and Warmuth studied the problem of approximating an unknown target distribution P by a probabilistic automaton [2]. The representation class of the hypotheses is the class of probabilistic automata or HMMs. More precisely, hypotheses are represented by PNFA with no final probabilities and the target distributions are defined on Σ^n. In this learning model an *automaton constraint* is also given to the learning algorithm.

[9] [17]:

$$2 \ln 2 \sqrt{D_{KL}(P, \hat{P})} \geq L_1(P, \hat{P}) = \sum_u |P(u) - \hat{P}(u)|$$

[10] [9]:

$$D_{KL}(P, \hat{P}) \geq D_H(P, \hat{P}) = \sum_u |\sqrt{P(u)} - \sqrt{\hat{P}(u)}|^2$$

Definition 22 *An automaton constraint is a 4-tuple $C = \langle \Sigma, Q, I, T \rangle$ where Σ is an alphabet, Q is a state set, $I \subseteq Q$ is a set of potentially initial states, and $T \subseteq Q \times \Sigma \times Q$ is a set of potential transitions. A PNFA satisfies the constraint C if its alphabet is Σ, its state set is Q, any initial state of its support automaton belongs to I, and any transition of its support automaton belongs to T. The* constraint size *is defined as $|C| = |I| + |T|$. It corresponds to the number of probabilities to estimate. The constraint is* null *if $I = Q$ and $T = Q \times \Sigma \times Q$.*

Given an automaton constraint the learning problem can be formulated as follows.

Definition 23 *An automaton constraint class $\mathcal{C}$ is* trainable *if there exists a learning algorithm satisfying the following conditions. For any constraint $C \in \mathcal{C}$, the algorithm receives C, an iid sample S_P drawn from an unknown distribution P defined on Σ^n, a precision parameter $\epsilon > 0$, and a confidence parameter δ, $0 < \delta \leq 1$. The algorithm outputs, with probability at least $1 - \delta$, a hypothesis $\hat{P}$ satisfying*

$$D_{KL}(P, \hat{P}) - D_{KL}(P, P_{min}(C)) \leq \epsilon,$$

provided $D_{KL}(P, P_{min}(C))$ is finite and provided the sample size m is greater than a minimal size m_{min}. $P_{min}(C)$ denotes the distribution generated by a probabilistic automaton satisfying the constraint C and presenting the minimal divergence with respect to the target P.
The class $\mathcal{C}$ is polynomially trainable *if any constraint $C \in \mathcal{C}$ is trainable with a minimal sample size m_{min} being a polynomial function of $\frac{1}{\epsilon}, \frac{1}{\delta}, n, |C|$, and if the time complexity of the learning algorithm is a polynomial function of the sample size.*

Note that if the target distribution P can be generated by a probabilistic automaton satisfying C then $D_{KL}(P, P_{min}(C)) = 0$, and the condition to be satisfied by $\hat{P}$ is to be an ϵ-good hypothesis with respect to P.

Learning a probabilistic automaton under a null constraint is equivalent to the problem of estimating probabilities when the alphabet and the number of states of the hypothesis are given. Determining whether a class of automaton constraints is polynomially trainable is therefore equivalent to determining whether there exists a polynomial algorithm to best estimate the probabilities of an automaton belonging to the constraint class. Given a sample $S_P = \{u_1, \ldots, u_m\}$ made of m words of Σ^n drawn independently according to the target distribution P, and given a hypothesis $\hat{P}$, the likelihood $L_{\hat{P}}(S_P)$ of the sample is defined as

$$L_{\hat{P}}(S_P) = \prod_{i=1}^{m} \hat{P}(u_i) \tag{4}$$

If the hypothesis $\hat{P}$ is considered as a model $\hat{M}$ belonging to a model class $\mathcal{M}$, the sample likelihood $L_{\hat{P}}(S_P)$ can be seen as $P(S_P|\hat{M})$, which is the probability of the sample given the model $\hat{M}$.

Definition 24 *The maximum likelihood problem is ϵ-approximable if there exists a learning algorithm that, when given a constraint C and a sample S_P, outputs, with probability at least $\frac{1}{2}$, a hypothesis $\hat{P}$ respecting C and satisfying*

$$\frac{L_{P_{max}(C)}(S_P)}{L_{\hat{P}}(S_P)} \leq 1 + \epsilon \tag{5}$$

where $P_{max}(C)$ denotes the distribution generated by a probabilistic automaton respecting C and assigning the maximal likelihood to the sample S_P.

The links between learning under a known constraint and estimating model parameters according to maximum likelihood are presented in section 5.4.

4.3 Identification in the limit with probability 1

Identification in the limit was introduced by Gold as a learning model in a non-probabilistic setting [30]. An adapted version of this model for language identification from stochastic examples was proposed by Angluin [3]. Identification of the support of probabilistic automata is described hereafter.

Definition 25 *A probabilistic automaton class $\mathcal{A}$ is* identifiable in the limit with probability 1 *if there exists a learning algorithm satisfying the following conditions. For any automaton $A \in \mathcal{A}$, the algorithm receives an infinite sequence of samples $S_1 \subseteq S_2 \subseteq \ldots$, each sample being drawn according to the same distribution P_A. The algorithm produces a sequence of hypotheses $\hat{P}_1, \hat{P}_2, \ldots$ such that, with probability 1, there is a finite index k^* from which, for any $k \geq k^*$, the support automaton $\hat{P}_k$ is the support automaton of A.*

This learning model concentrates on the exact identification, in finite time, of a support automaton. There is no required bound on the error before identification nor on the time complexity of the learning process. This observation might explain why the PAC model described in section 4.1 is generally preferred. Nevertheless the ALERGIA algorithm described in section 6.2.2, and several of its variants, have been proved to converge according to definition 25 [13,14,18].

In the identification in the limit framework, a sample S_c is called *characteristic* if the convergence is guaranteed for any sample S including S_c. Once the support has been identified, learning is reduced to correct estimation of the probabilities as defined, for instance, in section 4.2.

4.4 Bayesian learning and MDL principle

We present in this section the Bayesian learning framework, which does not constitute a learning model as described before. Yet this framework is frequently used in the literature, in particular in the context of HMM induction. The algorithms presented in sections 6.2.4, 6.2.5 and 6.4 are Bayesian learning techniques.

Let a probabilistic automaton $\hat{A}$ be a particular model belonging to an automaton class $\mathcal{A}$. $P(\hat{A})$ denotes the *prior* probability [11] of the model $\hat{A}$ in the class $\mathcal{A}$. $P(S|\hat{A})$ denotes the likelihood of the sample S given an automaton $\hat{A}$. The maximum *a posteriori* (MAP) learning principle consists in choosing the hypothesis $\hat{A}_{MAP}$ that maximizes $P(\hat{A}|S)$, which is the *posterior* probability of the model $\hat{A}$ given the sample S:

$$\hat{A}_{MAP} = \operatorname*{argmax}_{\hat{A} \in \mathcal{A}} P(\hat{A}|S) = \operatorname*{argmax}_{\hat{A} \in \mathcal{A}} P(S|\hat{A})P(\hat{A}) \tag{6}$$

where the second equality results from applying Bayes rule. Bayesian learning aims at selecting the model that maximizes a trade-off between sample likelihood and prior probability. When all models in the class are considered equally likely, Bayesian learning seeks for a maximum likelihood model. The link between trainability of an automaton given a constraint (see section 4.2)

[11] There is an implicit assumption that the prior probability of any model in the class $\mathcal{A}$ is well defined.

and parameter estimation following maximum likelihood is clarified by theorem 6 in section 5.4. Algorithms for maximum likelihood estimation are described in section 6.1.

Under certain hypotheses MAP learning is equivalent to the minimum description length (MDL) learning principle [47]. More precisely, $\hat{A}_{MAP}$ can be equivalently defined as follows:

$$\hat{A}_{MAP} = \underset{\hat{A}\in\mathcal{A}}{\operatorname{argmin}} - \log_2 P(S|\hat{A}) - \log_2 P(\hat{A}) \tag{7}$$

The term $-\log_2 P(S|\hat{A})$ is the description length of the sample S when an optimal code is chosen for encoding this sample given the model $\hat{A}$. The term $-\log_2 P(\hat{A})$ is the description length of the model $\hat{A}$ when an optimal code is chosen for encoding this model. The MDL principle recommends to select the hypothesis (the model) that minimizes the sum of both description lengths. The model $\hat{A}_{MAP}$ is therefore an MDL solution under optimal encoding schemes.

5 Learning results

We present in this section learning results for several distribution classes according to the learning models described in section 4.

5.1 Non-learnability of PDFA with an evaluator

The class $\mathcal{P}DFA_{2,r(n)}$ denotes distributions defined over Σ^n, with $|\Sigma| = 2$, that can be generated by PDFA without final probabilities, and having a number of states bounded by a polynomial $r(n)$. In this case n is the only parameter defining the size of the concept to be learned.

Kearns et al. introduce the distinction[12] between *generators* and *evaluators* for a distribution P. A generator for a distribution P takes as input a sequence of truly random bits and outputs an observation drawn according to P. An evaluator for a distribution P takes as input an observation and outputs the probability of this observation according to P. We look generally for learning algorithms producing evaluators since, once an evaluator has been learned, the probability of any new observation can be computed.

Theorem 1 *Under the noisy parity assumption*[13] *, the class* $\mathcal{P}DFA_{2,r(n)}$ *is not efficiently learnable [39].*

This result is independent of the representation class of the hypothesis $\hat{P}$ but it is assumed that the learning algorithm outputs an evaluator for the distribution $\hat{P}$.

[12] This distinction should not be confused with the distinction between generators and acceptors introduced at the end of section 2.3. For instance, a PDFA is both an evaluator and a generator in the sense defined in the current paragraph.

[13] There is a constant $0 < \eta < \frac{1}{2}$ such that there is no efficient algorithm for learning parity functions under the uniform distribution in the PAC model with classification noise rate η.

5.2 Learnability of μ-distinguishable acyclic automata

Ron et al. study the class of μ-distinguishable acyclic PDFA (APDFA), which forms a particular subclass of PDFA with final probabilities. The transition graph associated to the support automaton of an APDFA contains no cycle. The support language is therefore finite. The *depth* of an APDFA is the length of the longest path from the initial state to a final state.

Definition 26 *Let $A = \langle \Sigma, Q, \phi, \iota, \tau \rangle$ be a probabilistic automaton and let μ be a parameter, $0 \leq \mu \leq 1$. A pair of states q_1 and q_2 from Q is μ-distinguishable if there exists a word $u \in \Sigma^*$ such that $|P_{A_{q_1}}(u) - P_{A_{q_2}}(u)| \geq \mu$. The automaton A is μ-distinguishable if any pair of distinct states is μ-distinguishable.*

$\mathcal{APDFA}_{\mu,|Q|,|\Sigma|}$ denotes the class of acyclic μ-distinguishable PDFA with $|Q|$ states and defined on an alphabet of size $|\Sigma|$.

Theorem 2 *The class $\mathcal{APDFA}_{\mu,|Q|,|\Sigma|}$ is efficiently learnable when the parameter μ, $0 < \mu \leq 1$, is known by the learner. The learning algorithm outputs an ϵ-good hypothesis in time polynomial in $|Q|, |\Sigma|, \frac{1}{\mu}, \frac{1}{\epsilon}, \log \frac{1}{\delta}$ [54].*

Leveled APDFA is the hypothesis representation class chosen in this case. In a leveled APDFA, the level of a state q is defined as the unique length of any path leading from the initial state to q. For any APDFA having $|Q|$ states and depth d, there exists an equivalent leveled APDFA having $\mathcal{O}(|Q| \times d)$ states. The learning algorithm for this representation class is further detailed in section 6.2.3.

5.3 Learnability of probabilistic automata with variable memory length

Ron et al. introduced the class of Probabilistic Finite Suffix Automata of order L (L-PFSA) [55]. L-PFSA form a proper subclass of PDFA equivalent to variable order Markov chains, the maximal order of which is fixed to a positive integer L. L-PFSA do not include final probabilities and generate distributions over Σ^n, $n > 0$.

Ron et al. proposed a learning model for PFSA, slightly adapted from the PAC model described in section 4.1. The distance measure between a target distribution P and a hypothesis $\hat{P}$ is defined here as the *per symbol* Kullback-Leibler divergence:

$$\frac{1}{n} D_{KL}(P, \hat{P}) = \frac{1}{n} \sum_{u \in \Sigma^n} P(u) \log \frac{P(u)}{\hat{P}(u)} \tag{8}$$

This normalized distance is independent of the length n of the words on which it is computed. $\mathcal{PFSA}_{L,|Q|,|\Sigma|}$ denotes the class of L-$PFSA$ with $|Q|$ states and defined on an alphabet of size $|\Sigma|$.

Theorem 3 *The class $\mathcal{PFSA}_{L,|Q|,|\Sigma|}$ is efficiently learnable when the order L is known by the learner [55].*

Prediction Suffix Trees [14] (PST) is the hypothesis representation class. The learning algorithm

[14] Prediction Suffix Trees, also referred to as *Probabilistic Suffix Trees*, are formally defined in [55].

described in section 6.3.2 returns a PST that is an ϵ-good hypothesis the size of which is in $\mathcal{O}(L \times |Q| \times |\Sigma|)$.

5.4 Trainability of probabilistic automata

The problem of approximating an unknown distribution by a probabilistic automaton is studied by Abe and Warmuth [2]. The learning algorithm receives an automaton constraint the size of which defines the number of parameters to be estimated. The main results in this model are described below.

Theorem 4 *The class of PDFA constraints is polynomially trainable [2].*

In other words, finding a probabilistic automaton that best approximates a target distribution and satisfies a given deterministic constraint, is feasible in polynomial time.

Theorem 5 *The class of 2-states null PNFA constraints is not polynomially trainable, unless **RP=NP** [2].*

Note that this result is due to a time complexity being an exponential function of the constraint size. As in this case the number of states is fixed, the problem complexity actually depends exponentially on the alphabet size.

Theorem 6 *A constraint class is polynomially trainable if and only if, for any constraint C in the class and given a sample containing m words over Σ^n, the maximum likelihood problem is ϵ-approximable by an algorithm running in random time polynomial in $\frac{1}{\epsilon}, |C|, n, m$ [2].*

Since PNFA are equivalent to HMMs (see section 3), theorems 5 and 6 imply that estimating the parameters of a HMM so as to maximize the sample likelihood is not feasible in polynomial time. An open question is to determine particular subclasses of HMMs, more general than those equivalent to PDFA, for which a better complexity can be obtained. Note that the EM algorithm[15] outputs a *locally* optimal solution to the maximum likelihood parameter estimation for HMMs [10,19,53]. Maximum likelihood estimation is further detailed in section 6.1.

5.5 Identification in the limit of probabilistic automata

Carrasco and Oncina study the problem of identifying the support of PDFA. The main result in this model is summarized by the following theorem.

Theorem 7 *The class $\mathcal{PDFA}$ is identifiable in the limit with probability 1 [14].*

Let $|Q|$ denote the number of states of the target automaton and let m denote the size of the sample received at a given step of the identification process. At each step, the learning algorithm has a time complexity in $\mathcal{O}(m \times |Q|^2 \times |\Sigma|)$. Identification is guaranteed in a finite number of steps.

Carrasco and Oncina give a lower bound on the size of a characteristic sample [14]. This bound depends on the difficulty of distinguishing pairs of states of the target automaton by a common

[15] The EM algorithm is also called *Forward-Backward* or *Baum-Welch* algorithm in this context.

suffix of sufficiently different probability. In other words, the difficulty of learning depends on the distinguishability of pairs of states (see definition 26). Once the support has been learned, the problem of estimating the probabilities of a PDFA is easily solved[16] (see section 6.1).

Esposito et al. study the identification of probabilistic residual finite state automata (PRFA). The $\mathcal{P}RFA$ class includes properly the $\mathcal{P}DFA$ class and is strictly included in the $\mathcal{P}NFA$ class [25].

Theorem 8 *The $\mathcal{P}RFA$ class is identifiable in the limit with probability 1 if the learning algorithm has access to the exact probabilities of the words in the sample [25].*

The proposed learning algorithm runs in time polynomial in the sample size. This result is preliminary however as it relies on the assumption of knowing the probabilities of the sample words according to the target distribution. An open question is how to extend this result when these probabilities have to be estimated.

5.6 Learnability of probabilistic concepts

The results presented here do not concern the learning of automata that are probabilistic *acceptors* (see the discussion at the end of section 2.3), as we focus on models directly related to HMMs. Probabilistic acceptors form a particular case of *probabilistic concepts*, which randomly map an input set X to an output set Y. Probabilistic acceptors define *conditional* probability distributions $P(Y|X)$, instead of the unconditional distributions considered in the present paper. Learning samples for probabilistic acceptors are made of elements of $X \times Y$. The data are randomly drawn according to a fixed distribution over X and probabilistically labeled according to the distribution $P(Y|X)$. More details on the learning of probabilistic concepts are given in [66,1,72,38].

6 Induction algorithms

In this section we present various algorithms for learning probabilistic automata and HMMs. In each case we use the representation class for which the algorithm was described originally. A change of representation, in particular from probabilistic automata to HMMs (or conversely), can be performed following the results of section 3.

We recall briefly some well known techniques to estimate probabilities for these models when the topology is known. The topology, also called the structure of the model, can be seen as an automaton learning constraint (see definition 22). Next we concentrate on various induction algorithms for these models, combining the problems of structure induction and probability estimation.

[16] Note that the ALERGIA algorithm described in section 6.2.2 learns the support automaton and estimates the probabilities simultaneously.

Given an HMM $M = \langle \Sigma, Q, A, B, \iota \rangle$ for which a constraint $\langle \Sigma, Q, I, T \rangle$ is known, the problem is to estimate its probabilities from a sample. Maximum likelihood is the most popular estimation criterion in this context (see section 4.4).

Let $\lambda = \{A, B, \iota\}$ denote the set of parameters to be estimated and M_λ the corresponding model. Let $S = \{u_1, \ldots, u_m\}$ denote a learning sample. The problem consists in finding $\hat{\lambda}$ that maximizes the sample likelihood:

$$\hat{\lambda} = \operatorname*{argmax}_{\lambda} P(S|M_\lambda) = \operatorname*{argmax}_{\lambda} \prod_{i=1}^{m} P(u_i|M_\lambda) \tag{9}$$

Baum-Welch algorithm uses an iterative procedure producing a solution corresponding to a local maximum [11,10]. This algorithm can be seen as a particular case of the EM algorithm [19]. At each step the expected likelihood of the sample is computed given current parameter estimates (*Expectation step*). The parameter estimates are then updated while increasing the sample likelihood (*Maximization step*).

The probability of generating the word u_i by the model can be formulated as follows

$$P(u_i|M_\lambda) = \sum_{\nu \in Q^*} P(u_i, \nu|M_\lambda) = \sum_{\nu \in Q^*} P(u_i|\nu, M_\lambda) P(\nu|M_\lambda) \tag{10}$$

where Q^* denotes the set of possible state sequences, that is the set of possible paths through the underlying structure, generating u_i. Direct computation of this probability has a time complexity in $\mathcal{O}(|u_i||Q|^{|u_i|})$. The Baum-Welch algorithm uses the so-called Forward and Backward recurrences in order to reduce this complexity to $\mathcal{O}(|u_i||Q|^2)$.

The Viterbi algorithm [70,26] computes an approximation of the generation probability based on a single path [17] of maximal probability $P(u_i|M_\lambda) \approx P(u_i, \nu_{max}|M_\lambda)$ with

$$\nu_{max} = \operatorname*{argmax}_{\nu \in Q^*} P(u_i, \nu|M_\lambda) \tag{11}$$

$\forall a \in \Sigma, \forall q \in Q$, the estimate $\hat{B}(q, a)$ of the emission probability of the letter a on state q is given by

$$\hat{B}(q, a) = \begin{cases} \frac{C(q,a)}{C(q)} & \text{, if } C(q) > 0, \\ 0 & \text{, otherwise.} \end{cases} \tag{12}$$

where $C(q, a)$ denotes the number of times the letter a was emitted on state q along the path ν_{max} for each word of the sample S, and $C(q) = \sum_{a \in \Sigma} C(q, a)$. The other parameters are estimated in a similar way. More details about Baum-Welch and Viterbi algorithms are presented in [52,37,53,36,24].

Note that once the HMM parameters are known, the Forward recurrence can be used to compute efficiently the probability of generating any new word u by the HMM. Similarly the Viterbi algorithm returns the path ν_{max}, which is a maximal probability state sequence generating this

[17] There is no guarantee that the maximal probability state sequence is generally unique. The Viterbi algorithm returns one such state sequence.

word. In other words, this algorithm provides a maximal probability alignment between each letter of the word u and the model states.

All the results presented here can be applied to PNFA as they are equivalent to HMMs (see section 3). In the particular case of PDFA, the estimation problem is simplified as there is at most one generating path for each word of the learning sample. This unique generating path is also the Viterbi path. In this case, the maximum likelihood estimate of a transition probability[18] is given by:

$$\hat{\phi}(q,a) = \begin{cases} \frac{C(q,a)}{C(q)} & \text{, if } C(q) > 0, \\ 0 & \text{, otherwise.} \end{cases} \tag{13}$$

where $C(q,a)$ denotes the number of times the transition (q,a) was used while generating S.

When M is a PDFA, the time complexity of the exact computation of $P(u_i|M_\lambda)$ is in $\mathcal{O}(|u_i|)$. A similar simplified computation can be derived for the more general class of non ambiguous probabilistic context-free grammars [16]. The general case of estimating the parameters of ambiguous probabilistic context-free grammars can be solved by the Inside-Outside algorithm [6,40,41].

6.2 State merging induction algorithms

In this section we describe several induction algorithms that generalizes the learning sample by merging states of a trivial PDFA built on this sample. This initial PDFA, often called the *Probabilistic Prefix Tree Acceptor* (PPTA), is presented in section 6.2.1. Several state merging algorithms are described next.

6.2.1 Probabilistic prefix tree acceptor and quotient automaton

Given a learning sample S, the *prefix tree acceptor*[19] is a DFA that only generates S. Its state set is the set of prefixes of words belonging to S, resulting in a tree shaped automaton. $PPTA(S)$ denotes the probabilistic prefix tree acceptor. It is a PDFA with finite support S as defined in proposition 3, where the distribution ψ considered is the sample distribution: if $C(u)$ denotes the count of the word u in the learning sample S, which is a multi-set, the sample distribution is defined as $\psi(u) = \frac{C(u)}{\sum_u C(u)}$.

Let A denote a PNFA the state set of which is Q. Assume the support language of A includes a sample S. Let A_Π denote a PNFA derived from A with respect to the partition Π of Q. A_Π is called a *quotient automaton* of A. It is obtained by merging states of A belonging to the same subset in Π. When a state q (resp. q') from A_Π results from the merging of the states $\{q_1, \ldots, q_k\}$

[18] We adopt a simplified notation for PDFA as, for any pair (q,a), there is at most one state q' such that $\phi(q,a,q') > 0$. In the sequel, this probability is simply denoted by $\phi(q,a)$.

[19] In a non-probabilistic setting, the same automaton can be seen as an acceptor or a generator of words belonging to some regular language. The probabilistic version we consider here is a PDFA seen as a probabilistic *generator*. Probabilistic Prefix Tree Generator would therefore be a better name but it is not commonly used in the literature.

(resp. $\{q'_1, \ldots, q'_l\}$) from A the following equalities must hold

$$\forall a \in \Sigma, C(q, a, q') = \sum_{i=1}^{k} \sum_{j=1}^{l} C(q_i, a, q'_j) \tag{14}$$

In the particular case of A_Π being a PDFA, the previous equalities are simply written as

$$\forall a \in \Sigma, C(q, a) = \sum_{i=1}^{k} C(q_i, a) \tag{15}$$

Figure 11 presents an example of $PPTA(S)$ built on the sample $S = \{a, aa, b, b, b\}$ and its quotient automaton obtained from the partition $\Pi = \{\{\varepsilon, a\}, \{aa\}, \{b\}\}$.

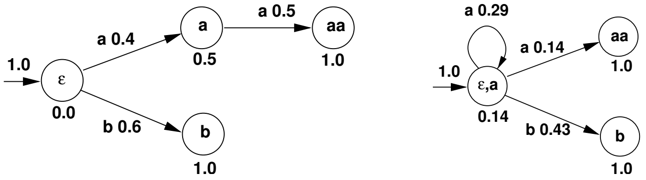

Figure 11. A probabilistic prefix tree automaton and a quotient automaton.

State merging is a generalization operation since the relation between support languages is $L(\underline{A}) \subseteq L(\underline{A_\Pi})$. The associated probability distributions differ whenever $L(\underline{A}) \subsetneq L(\underline{A_\Pi})$. The set of all probabilistic automata that can be derived from $PPTA(S)$ by merging some states, which is the set of quotient automata of $PPTA(S)$, defines a search space of automata generalizing the learning sample. This search space includes in particular all PDFA that can be derived from $PPTA(S)$. Properties of the same search space considered in a non-probabilistic setting are described in [49,23].

```
Input: A learning sample $S$; a precision parameter $\mu$
Output: A probabilistic automaton
$A \longleftarrow PPTA(S)$                                    // Initial solution
while (Stopping criterion not satisfied) do
    $(q, q') \longleftarrow$SelectStates($A$)                 // Select state pair
    if Compatible($q, q', \mu$) then
        $A \longleftarrow$Update ($A, q, q'$)                 // Update current solution
return $A$
```

Figure 12. A generic induction algorithm using state merging

Figure 12 depicts a generic learning algorithm using state merging. A state pair is first selected from $PPTA(S)$ and this pair is a candidate for merging. The function `SelectStates` defines the order in which candidate state pairs are considered. The function `Compatible` tests whether two states should be merged according to some statistical criterion and a precision parameter μ. If the candidate state pair is compatible, the current automaton is updated by merging q and q', and, possibly, some additional states. Candidate state pairs are considered for merging till some stopping criterion is met. The merging algorithms described in the next sections can be formulated according to specific definitions of the functions `SelectStates`, `Compatible`, `Update`, and the stopping criterion.

6.2.2 The ALERGIA algorithm

The ALERGIA algorithm [13] induces a PDFA from a learning sample. The states of $PPTA(S)$ are associated to prefixes which may be sorted according to the standard order on strings[20]. Candidate states for merging are considered in this order. `SelectStates` returns state pairs made of a given state and each of its predecessors following the same order. For example, the first candidate state pairs on a two letter alphabet can be[21] $(a, \varepsilon), (b, \varepsilon), (b, a), (aa, \varepsilon), (aa, a), (aa, b), \ldots$ The stopping criterion is defined as the end of the enumeration of prefix pairs actually present in $PPTA(S)$. $\mathcal{O}(n^2)$ candidate state pairs are therefore checked for merging compatibility, where n denotes the number of states of $PPTA(S)$.

The function `Compatible` implements a compatibility measure derived from the Hoeffding bound [33]. Formally, two states q and q' are μ-compatible $(0 < \mu \leq 1)$ if the two following conditions hold:

$$\text{(16.1)} \qquad \left|\frac{C(q,a)}{C(q)} - \frac{C(q',a)}{C(q')}\right| < \sqrt{\frac{1}{2}\ln\frac{2}{\mu}}\left(\frac{1}{\sqrt{C(q)}} + \frac{1}{\sqrt{C(q')}}\right), \ \forall a \in \Sigma$$

$$\text{(16.2)} \qquad \delta(q,a) \text{ and } \delta(q',a) \text{ are } \mu\text{−compatible}, \ \forall a \in \Sigma$$

Condition (16.1) defines the compatibility between each pair of transitions outgoing respectively from state q and q'. The same condition must hold for final probability estimates obtained by replacing $C(q, a)$ with $C(q, \#)$ (resp. $C(q', a)$ with $C(q', \#)$), where # is a special *end-of-word* symbol. Condition (16.2) requires the compatibility to be recursively satisfied for every pair of successors of these states[22].

The `Update` function merges two compatible states q and q', and, recursively, all their respective successors in order to eliminate non-determinism in the underlying structure. Figure 13 depicts an execution example of the `Update` function. The temporary solution is represented at the top and probabilities are left out here for clarity. The state pair (ba, b) is assumed to satisfy the compatibility measure. These states are merged resulting in a new quotient automaton, which in this case is structurally non-deterministic. Subsequent merging steps are then performed to eliminate non-determinism. This results in the automaton, depicted at the bottom of the figure, which is guaranteed to be a PDFA. This recursive merging operation is sometimes called *determinization by merging.*

[20] According to the standard order denoted $<$, the first strings on the alphabet $\Sigma = \{a, b\}$ are $\varepsilon < a < b < aa < ab < ba < bb < aaa < \ldots$

[21] The candidate state pairs actually considered are only those present in $PPTA(S)$.

[22] If one successor state, say q', is undefined for the transition function δ is partial, condition (16.1) can be rewritten for the other successor q as follows: $\left|\frac{C(q,a)}{C(q)}\right| < \sqrt{\frac{1}{2}\ln\frac{2}{\mu}}(\frac{1}{\sqrt{C(q)}})$, $\forall a \in \Sigma$.

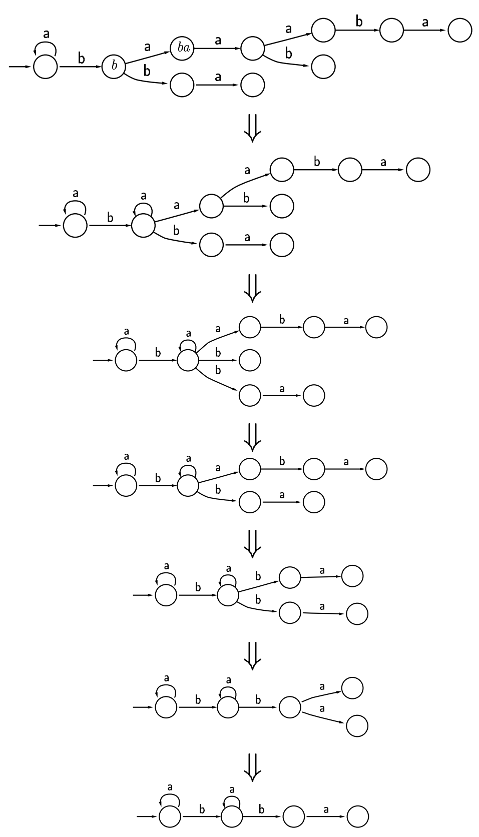

Figure 13. An `Update` example including determinization by merging.

The class $\mathcal{P}DFA$ can be identified in the limit with probability one using the ALERGIA algorithm. A slightly modified algorithm, called RLIPS, was proposed later with a reformulated proof of convergence [14]. Finally note that, in the case of small finite samples, state pairs with very low counts can be wrongly considered compatible. This was observed experimentally by Young-Lai and Tompa who introduced some refinements to the compatibility measure [73].

6.2.3 Learning acyclic PDFA

Ron et al. proposed an algorithm for learning acyclic PDFA [56]. It is similar to ALERGIA but for the definitions of `SelectStates` and `Compatible`. Candidate state pairs are restricted to states sharing the same level in $PPTA(S)$. These states are associated with prefixes of the same length. Thus the quotient automata are acyclic. State pairs are considered in increasing level order, and, for a given level, in arbitrary order. In addition, a given state can be selected only if its count is greater or equal to a predefined threshold. All low count states belonging to the same level are eventually merged in a single state, irrespective of their compatibility. On the other hand, two states q and q' are μ-compatible if

$$\left| \frac{C(q, a\Sigma^*)}{C(q)} - \frac{C(q', a\Sigma^*)}{C(q')} \right| \leq \frac{\mu}{2},\ \forall a \in \Sigma \tag{17}$$

where $C(q, a\Sigma^*)$ denotes the count of the suffixes of state q starting with letter a in the current solution. These counts can be computed efficiently in $PPTA(S)$ and updated whenever states are merged. Transition probabilities are finally smoothed by correcting the maximum likelihood estimates as follows

$$\hat{\phi}(q,a) = \frac{C(q,a)}{C(q)}(1-(|\Sigma|+1)\phi_{min}) + \phi_{min} \tag{18}$$

where ϕ_{min} denotes the minimal probability assigned to any transition.

The class of μ-distinguishable APDFA is PAC learnable using the proposed algorithm (see section 5.2). This interesting result is limited however by the fact that the induced support languages are necessarily finite.

6.2.4 The MDI algorithm

The MDI algorithm [64] differs from ALERGIA in the definition of the `Compatible` function. This algorithm aims at inducing PDFA while trading off minimal divergence from the training sample distribution and minimal size. It can be considered as a Bayesian learning method (see section 4.4). Indeed a possible solution with null divergence is $PPTA(S)$. It is also a maximum likelihood model built from the learning sample. On the other hand, favoring small automata, or equivalently automata derived from $PPTA(S)$ with a large number of merging operations, corresponds to an increased *prior* probability associated to a reduced automaton size. Trading off both effects is thus equivalent to maximizing the *posterior* probability of the model given the training data.

Assume $A_0 = PPTA(S)$, A_1 is a temporary solution and A_2 is a tentative new solution that can be derived from A_1. In other words, A_2 can be obtained from A_1 by merging some candidate state pair q and q', and, possibly, some additional states according to the determinization by merging operation. States q and q' are μ-compatible if

$$\frac{D(A_0||A_2) - D(A_0||A_1)}{|A_1| - |A_2|} < \mu \tag{19}$$

In other words, A_2 is the new temporary solution if the divergence increment relative to the size reduction, that is, the reduction of the number of states, is less than μ. Note that, when the *prior* probability of A_i is defined as $P(A_i) = 2^{-|A_i|}$, the denominator of expression (19) is equivalent to the log ratio of the *priors*: $\log_2 \frac{P(A_2)}{P(A_1)}$. The divergence increment can be efficiently computed as explained below.

Let $PPTA(S) = A_0 = \langle \Sigma, Q^0, \phi^0, \iota^0, \tau^0 \rangle$ and let $A_0/\pi_{01} = A_1 = (\Sigma, Q^1, \phi^1, \iota^1, \tau^1)$ be a deterministic quotient automaton of A_0. By definition of a quotient automaton, each state q_i in A_0 exactly corresponds to one state $q_{\underline{i}} \triangleq B_{\Pi_{01}}(q_i)$ in A_1. In other words, $B_{\Pi_{01}}(q_i)$ denotes the subset of the partition Π_{01} to which the state q_i belongs. Let c_i denote the probability of reaching q_i from the tree root. The divergence between A_0 and A_1 can be computed as:

$$\begin{aligned} D(A_0||A_1) &= \sum_{q_i \in Q_0} \sum_{a \in \Sigma \cup \{\#\}} c_i \; \phi_0(q_i,a) \log \frac{\phi_0(q_i,a)}{\phi_1(q_{\underline{i}},a)} \\ &= - \sum_{q_i \in Q_0} \sum_{a \in \Sigma \cup \{\#\}} c_i \; \phi_0(q_i,a) \log \phi_1(q_{\underline{i}},a) - H(A_0) \end{aligned} \tag{20}$$

where $H(A_0)$ denotes the entropy of A_0. The divergence $D(A_0||A_1)$ is always finite in this case as $\phi_1(q_{\underline{i}},a) \neq 0$ if $\phi_0(q_i,a) \neq 0$. Let $A_2 = A_1/\pi_{12}$ be a deterministic quotient automaton of A_1. By construction, A_2 is also a quotient automaton of A_0 for some partition π_{02}. Thus the divergence increment can be computed as follows:

$$D(A_0||A_2) - D(A_0||A_1) = \sum_{q_i \in Q_{012}} \sum_{a \in \Sigma \cup \{\#\}} c_i \, \phi_0(q_i, a) \log \frac{\phi_1(q_{\underline{i}}, a)}{\phi_2(q_{\underline{i}}, a)} \tag{21}$$

where $Q_{012} = \{q_i \in Q_0 \; |B_{\pi_{01}}(q_i) \neq B_{\pi_{02}}(q_i)\}$ denotes the set of states in A_0 that have been merged to derive A_2 from A_1.

There is no existing proof of convergence of MDI with respect to learning models described in section 4. Empirical results in the domain of language model construction for the ATIS travel information task [32] show however that MDI outperforms ALERGIA [64].

6.2.5 Bayesian HMM induction by state merging

Stolcke and Omohundro proposed an induction algorithm by Bayesian model merging [62,61]. This algorithm differs from the generic merging algorithm described at figure 12, since HMMs are chosen here as representation class, but several similarities can be observed.

The initial solution is a trivial HMM M_0 generating exactly the learning sample S. Each word u of S is associated to a specific path in M_0. The initial probability of the first state of each path is given by the relative frequency of u in S. Each path is made of $|u|$ states and the emission probability of each state is 1 for the corresponding letter. Each state is therefore initially assigned to a unique output symbol. Note that M_0 is a maximum likelihood model[23].

The *prior* probability of a model M_λ with parameters λ is defined as $P(M_\lambda) = P(M_s)P(\lambda|M_s)$, where $P(M_s)$ denotes the prior probability of the HMM structure and $P(\lambda|M_s)$ denotes the prior probability of the parameter values given the structure M_s. The structural prior is defined as $P(M_s) \propto e^{-|M|}$, where $|M|$ is the number of states of the HMM, producing a bias towards small models as for the MDI algorithm. Since transition and emission probabilities in a HMM can be seen as multinomial distributions, the parameter priors are assigned using a Dirichlet distribution. The effect of this prior is equivalent to having a number of additional virtual counts associated to each of the possible emissions and transitions. For example, the MAP estimate of the emission probability of letter a on state q is given by

$$\hat{B}(q, a) = \frac{C(q, a) + \alpha_e - 1}{\sum_{a \in \Sigma}[C(q, a) + \alpha_e - 1]} \tag{22}$$

where α_e is the virtual count associated to an emission. The virtual counts chosen in this case are making equal use of all potential emissions and transitions, adding bias towards uniform transition and emission probabilities.

Starting from the initial model, all state pairs are considered for merging and the resulting model that maximizes the posterior probability of the model structure is chosen. This probability incorporates a global prior weighting $\beta > 0$. The quantity to be maximized is defined as

$$\beta \log P(M) + \log P(S|M) \tag{23}$$

The merging step is iterated till a local maximum of the weighted posterior is found.

[23] The definition of the trivial model given in [61] slightly differs from our definition as it uses a distinct path associated to each repetition of a given word in S. However, equivalent states in this model can be merged to get M_0 without likelihood loss.

The time complexity of this algorithm is significantly larger than those of the algorithms described above. In particular, the number of state pairs considered *at each step* is in $\mathcal{O}(n^2)$, where n denotes the number of states of M_0. The total number of candidate state pairs is therefore in $\mathcal{O}(n^3)$. Moreover there is no analogue to the determinization by merging operation, which would reduce significantly the actual number of state pairs considered in practice.

Several heuristics are used here to decrease the number of candidate state pairs. For instance, early merging steps restrict candidates to state pairs having the same output symbol, while general merging is allowed in later stages. The evaluation of the posterior probabilities also includes several approximations. In particular, the model likelihood is computed using the Viterbi approximation described in section 6.1. It is also assumed that merging preserves the Viterbi paths.

One common problem observed in practice is that the stopping criterion is satisfied too early, as a single merging step could decrease the posterior model probability even though additional related steps might increase it. The stopping criterion is therefore modified to trigger only after a fixed number of steps have produced no improvement.

6.3 State splitting induction algorithms

State splitting is an induction technique opposite to state merging. A model with very few states (possibly a single one) is built initially. The topology of this initial model depends on the prior knowledge available. For instance, it can be a fully connected graph or a left-to-right structure, as in the case of HMMs used for acoustic modeling. Next, the model is iteratively specialized by splitting some states to best fit the training data.

An early approach using splitting is described in [20], where a stochastic regular grammar, equivalent to a PNFA, is iteratively specialized so as to maximize a Bayesian criterion. However, the enumerative search technique proposed has an exponential time complexity.

6.3.1 Successive state splitting

Successive state splitting was used to learn HMM topologies for allophone modeling [63]. An improved version of this technique is described in [48]. This approach was developed for continuous HMMs but the basic steps can be applied in the discrete case. An initial model topology is defined and parameters are estimated by maximum likelihood using the Baum-Welch algorithm (see section 6.1). At each step, a state is selected for splitting so as to maximize the expected log likelihood on a constrained subset of the parameters. Two types of splitting operations in a left-to-right structure are considered here. A contextual split replaces a given state by a pair of parallel states. A temporal split replaces a given state by a sequence of two states. Only those states affected by the splitting operation are considered while retraining the parameters with the Baum-Welch algorithm. The process is iterated till the likelihood gain falls below a given threshold. Note that the splitting operations considered here do not allow to induce cyclic models, unless cycles were already included in the initial topology. Additional criteria for state splitting are described in [60], including a chi-squared goodness of fit test, a cross-validation criterion, and an MDL stopping criterion.

6.3.2 *Prediction suffix trees learning*

Ron et al. proposed an induction technique for learning L-PFSA (see section 5.3), which form a subclass of PDFA equivalent to variable order Markov chains [55]. The representation class used for L-PFSA is the class of Prediction Suffix Trees (PSTs). Each state in a PST is associated to a specific suffix v and a conditional probability $P(a|v)$ of generating a letter a given the corresponding suffix v. The initial tree contains a single state associated to the empty suffix ε. Next, the tree is grown by considering increasingly larger suffixes up to a maximal length, which defines the model order. A state associated to the suffix v is created if there exists a letter a for which the maximum likelihood estimate $\hat{P}(a|v)$ satisfies the following conditions:

$$\hat{P}(a|v) \geq \eta \text{ and } \frac{\hat{P}(a|v)}{\hat{P}(a|v_{-1})} > 1 + \eta' \tag{24}$$

where η and η' are predefined thresholds, and v_{-1} denotes the longest suffix of v not equal to v. L-PFSA are PAC learnable using the proposed technique (see section 5.3).

6.4 *Structural induction by parameter estimation*

The MAP learning approach described in [12] folds HMM structure induction into parameter estimation. It uses an adapted version of the EM-algorithm where the M-step is modified so as to maximize the posterior probability of a model. An entropic posterior probability is obtained by combining an entropic prior with the sample likelihood. The entropic prior is adding bias towards sparse structures. The posterior defines a distribution over all possible model structures and parameterizations within a class. Starting from an initial model structure, for instance a fully connected graph, the MAP estimator drives irrelevant parameters to zero. Simple tests can then be performed to prune transitions and states while increasing the posterior probability of the model. Pruning accelerates training by removing parameters that would otherwise decay asymptotically to zero. MAP estimation combined with parameter pruning is therefore a structural induction technique.

Another approach using transition pruning was described in [68]. Starting from a fully connected HMM, the algorithm iteratively prunes transitions, and the resulting model likelihood is recomputed. The pruning process is iterated till the model likelihood does not decrease significantly. Note that this heuristic selection criterion could have been formalized in a Bayesian setting using a larger prior probability for a model with less transitions.

6.5 *Error-correcting induction techniques*

The ECGI algorithm uses error correcting techniques to induce an automaton structure [57]. The initial model only generates the first word of the learning sample. The model is then greedily adapted to best incorporate the rest of the sample. At each step, new states and transitions are added according to a minimal number of editing operations (substitution, deletion, and insertion) needed to accept the new words. These optimal editing operations are computed using dynamic programming. Maximum likelihood estimation of the model parameters can be computed simultaneously, and the editing costs can be defined according to updated estimates of probabilistic editing operations. Note that the final structure depends on the order in which the learning examples are considered. A very similar technique is described in [65].

7 Conclusions and perspectives

We studied the links between probabilistic automata and HMMs, showing that PDFA form a proper subclass of PNFA, and that PNFA and HMMs are equivalent. More precisely, there are two families of equivalent models according to whether or not final probabilities are included. In the former case, the models generate distributions over words of finite length, while, in the later case, distributions are defined over complete finite prefix-free sets. The equivalence between PNFA and HMMs allows to apply learnability results and induction algorithms developed in one formalism to the other.

Learnability results presented in section 5 illustrate the difficulty of learning probabilistic automata or HMMs. If the automaton structure is known then learning is reduced to a probability estimation problem. Looking for the model that globally maximizes the sample likelihood cannot be performed in polynomial time for the general classes of PNFA or HMMs. When the structure is unknown, learning in the PAC sense is not feasible even for PDFA defined on a 2-letter alphabet. Proper subclasses of PDFA are learnable: automata with bounded variable memory and μ-distinguishable acyclic PDFA. On the other hand, the class of PDFA is identifiable in the limit with probability 1 but this model does not bound the overall complexity of learning.

An open question is whether other interesting subclasses of PNFA are PAC learnable. Note also that non-learnability results mentioned here are related to automata with no final probabilities, and for which the learning objective is to minimize the divergence with respect to some target distribution. It would be worth to investigate learnability results for general PNFA including final probabilities. Alternatively, one could adopt a distance measure between distributions which would be easier to satisfy than the divergence. An interesting result along these lines was already mentioned by Fu [29]. It states that even a probabilistic context-free language can be approximated by a probabilistic finite support language, when the quadratic distance[24] is considered between both languages. Alternative criteria for approximating the maximum likelihood problem could be considered. In this context, can we characterize the locally optimal solution produced by the EM algorithm with respect to a solution that best approximates the global optimum?

While positive learnability results are difficult to obtain for the general class of PNFA or HMMs, several algorithms presented in section 6 can be used in practice. These learning algorithms usually restrict the learning to a particular model subclass, typically the class of PDFA. They do not always fit in with a learning model as described before but were generally developed in a Bayesian framework.

Finally, we believe that an important issue for practical applications with limited amounts of training data is the design of appropriate smoothing techniques for probabilistic automata. Approaches along these lines include symbol clustering [22], error-correcting smoothing [21] and back-off smoothing [44].

[24] The quadratic distance between distributions $P1$ and $P2$ is defined as: $D_Q(P_1, P_2) = \sum_{u \in \Sigma^*} (P_1(u) - P_2(u))^2$.

References

[1] N. Abe, J. Takeuchi, and M. K. Warmuth. Polynomial learnability of probabilistic concepts with respect to kullback-leibler divergence. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 277–289, 1991.

[2] N. Abe and M. Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9:205–260, 1992.

[3] D. Angluin. Identifying languages from stochastic examples. Technical Report YALEU/DCS/RR-614, Yale University, March 1988.

[4] L. Bahl, P. Brown, P. de Souza, and R. Mercer. Estimating Hidden Markov Models parameters so as to maximize speech recognition accuracy. *IEEE Transactions on Speech and Audio Processing*, 1(1):77–83, 1993.

[5] L. Bahl, F. Jelinek, and R. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2):179–190, 1983.

[6] J.K. Baker. Trainable grammars for speech recognition. In D. Klatt and J. Wolf, editors, *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550, 1979.

[7] V. Balasubramanian. *Equivalence and Reduction of Hidden Markov Models.* PhD thesis, MIT, 1993.

[8] P. Baldi and S. Brunak. *Bioinformatics: a machine learning approach.* MIT Press, 2nd edition, 2001.

[9] A.R. Barron and T.M. Cover. Minimum complexity density estimation. *IEEE Transactions on Information Theory*, pages 1034–1054, 1991.

[10] L. Baum. An inequality and associated maximization technique in statistical estimation for probalistic functions of Markov processes. *Inequalities*, 3:1–8, 1972.

[11] L. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occuring in the statistical analysis of probablistic functions of Markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[12] M.E. Brand. Structure learning in conditional probability models via an entropic prior and parameter extinction. *Neural Computation Journal*, 11(5):1155–1182, 1999.

[13] R. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *Grammatical Inference and Applications, ICGI'94*, number 862 in Lecture Notes in Artificial Intelligence, pages 139–150, Alicante, Spain, 1994. Springer Verlag.

[14] R. Carrasco and J. Oncina. Learning deterministic regular gramars from stochastic samples in polynomial time. *Theoretical Informatics and Applications*, 33(1):1–19, 1999.

[15] F. Casacuberta. Some relations among stochastic finite state networks used in automatic speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):691–695, 1990.

[16] R. Chaudhuri and A.N.V. Rao. Approximating grammar probabilities: Solution of a conjecture. *Journal of the Association for Computing Machinery*, 33(4):702–705, 1986.

[17] T. Cover and J. Thomas. *Elements of Information Theory.* John Wiley and Sons, New York, 1991.

[18] C. de la Higuera and F. Thollard. Identification in the limit with probability one of stochastic deterministic finite automata. In A. Oliveira, editor, *Grammatical Inference: Algorithms and Applications*, number 1891 in Lecture Notes in Artificial Intelligence, pages 15–24. Springer Verlag, 2000.

[19] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society Ser. B (methodological)*, 39:1–38, 1977.

[20] A. Van der Mude and A. Walker. On the inference of stochastic regular grammars. *Information and Control*, 38:310–329, 1978.

[21] P. Dupont and J.C. Amengual. Smoothing probabilistic automata: an error-correcting approach. In A. Oliveira, editor, *Grammatical Inference: Algorithms and Applications*, number 1891 in Lecture Notes in Artificial Intelligence, pages 51–64. Springer Verlag, 2000.

[22] P. Dupont and L. Chase. Using symbol clustering to improve probabilistic automaton inference. In *Grammatical Inference, ICGI'98*, number 1433 in Lecture Notes in Artificial Intelligence, pages 232–243, Ames, Iowa, 1998. Springer Verlag.

[23] P. Dupont, L. Miclet, and E. Vidal. What is the search space of the regular inference ? In *Grammatical Inference and Applications, ICGI'94*, number 862 in Lecture Notes in Artificial Intelligence, pages 25–37, Alicante, Spain, 1994. Springer Verlag.

[24] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis.* Cambridge University Press, 1998.

[25] Y. Esposito, A. Lemay, F. Denis, and P.Dupont. Learning probabilistic residual finite automata. In P. Adriaans, H. Fernau, and M. van Zaanen, editors, *Proceedings of the 6th International Colloquium on Grammatical Inference: Algorithms and Applications*, number 2484 in Lecture Notes in Artificial Intelligence, pages 77–91, Amsterdam, the Netherlands, September 2002. Springer Verlag.

[26] G.D. Forney. The Viterbi algorithm. *IEEE Proceedings*, 3:268–278, 1973.

[27] K.S. Fu. *Syntactic Methods in Pattern Recognition*, volume 112 of *Mathematics in Science and Engineering.* Academic Press, New-York, 1974.

[28] K.S. Fu and T.L. Booth. Grammatical inference: Introduction and survey, part 1. *IEEE Transactions on Systems, Man and Cybernetics*, 5:85–111, 1975.

[29] K.S. Fu and T.L. Booth. Grammatical inference: Introduction and survey, part 2. *IEEE Transactions on Systems, Man and Cybernetics*, 5:409–423, 1975.

[30] E.M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[31] R.C. Gonzalez and M. G. Thomason. *Syntacic Pattern Recognition, An Introduction.* Addition-Wesley, Reading, Massachusetts, 1978.

[32] L. Hirschman. Multi-site data collection for a spoken language corpus. In *Proceedings of DARPA Speech and Natural Language Workshop*, pages 7–14, Arden House, NY, 1992.

[33] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[34] J.J. Horning. *A Study of Grammatical Inference.* Ph. D. dissertation, Computer Science Department, Stanford University, Stanford, California, 1969.

[35] H. Huang and K.S. Fu. On stochastic context-free languages. *Information Sciences*, 3:201–224, 1971.

[36] F. Jelinek. *Statistical Methods for Speech Recognition.* MIT Press, Cambridge, MA, 1998.

[37] Y. Kamp. An introduction to the Baum and EM algorithms for maximum likelihood estimation. Technical Report 830, Institute for Perception Research, 1991.

[38] M. Kearns and R. Schapire. Efficient distribution-free learning of probabilistic concepts. In S.J. Hanson, G.A. Drastal, and R.L. Rivest, editors, *Computational Learning Theory and Natural Learning Systems, Volume I: Constraints and Prospect*, volume 1. MIT Press, 1994.

[39] M.J. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R.E. Schapire, and L. Sellie. On the learnability of discrete distributions. In *Proc. of the 25th Annual ACM Symposium on Theory of Computing*, pages 273–282, 1994.

[40] K. Lari and S.J. Young. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[41] K. Lari and S.J. Young. Applications of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 5:237–257, 1991.

[42] K.F. Lee. *Large Vocabulary Speaker Independent Continuous Speech Recognition : The SPHINX System.* Ph. D. dissertation, Computer Science Department, Carnegie Mellon University, 1988.

[43] E. Levin and R. Pieraccini. Planar hidden markov modeling: from speech to optical character recognition. In C.L. Giles, S.J. Hanton, and J.D. Cowan, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 731–738. Morgan Kauffman, 1993.

[44] D. Llorens, J.-M. Vilar, and F. Casacuberta. Finite state language models smoothed using n-grams. *International Journal of Pattern Recognition and Artificial Intelligence*, 16(3):275–289, 2002.

[45] F.J. Maryanski. *Inference of Probabilistic Grammars.* Ph. D. dissertation, University of Connecticut, 1974.

[46] L. Miclet. Grammatical inference. In H. Bunke and A. Sanfeliu, editors, *Syntactic and Structural Pattern Recognition: Theory and Applications*, volume 7 of *Series in Computer Science*, pages 237–290. World Scientific, Singapore, 1990.

[47] T. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[48] M. Ostendorf and H. Singer. Hmm topology design using maximum likelihood successive state splitting. *Computer Speech and Language*, 11:17–41, 1997.

[49] T. Pao and J. Carr. A solution to the syntactic induction-inference problem for regular languages. *Computer Languages*, 3:53–64, 1978.

[50] A. Paz. *Introduction to Probabilistic Automata.* Academic Press, 1971.

[51] A. Poritz. Hidden markov models: a guided tour. In *International Conference on Acoustic, Speech and Signal Processing*, pages 7–13, 1988.

[52] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[53] L. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition.* Prentice-Hall, 1993.

[54] D. Ron and R. Rubinfeld. Learning fallible deterministic finite automata. *Machine Learning*, 18:149–185, 1995.

[55] D. Ron, Y. Singer, and N. Tishby. Learning probabilistic automata with variable memory length. In *Proceedings of the Seventh Annual Conference on Computational Learning Theory*, New Brunswick, NJ, 1994. ACM Press.

[56] D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic automata. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, pages 31–40, Santa Cruz, CA, 1995. ACM Press.

[57] H. Rulot and E. Vidal. An efficient algorithm for the inference of circuit-free automata. In G. Ferratè, T. Pavlidis, A. Sanfeliu, and H. Bunke, editors, *Advances in Structural and Syntactic Pattern Recognition*, pages 173–184. NATO ASI, Springer-Verlag, 1988.

[58] Y. Sakakibara. Recent advances of grammatical inference. *Theoretical Computer Science*, 185(1):15–45, 1997.

[59] K. Seymore, A. McCallum, and R. Rosenfeld. Learning hidden markov model structure for information extraction. In *AAAI'99 Workshop on Machine Learning for Information Extraction*, pages 37–42, Orlando, Florida, 1999.

[60] B. Stenger, V. Ramesh, N. Paragios, F. Coetzee, and J. Buhmann. Topology free hidden markov models: Application to background modeling. In *Proceedings of the 8th International Conference on Computer Vision*, volume 1, pages 294–301, 2001.

[61] A. Stolcke. *Bayesian Learning of Probabilistic Language Models.* Ph. D. dissertation, University of California, 1994.

[62] A. Stolcke and S.M. Omohundro. Hidden markov model induction by bayesian model merging. In C.L. Giles, S.J. Hanton, and J.D. Cowan, editors, *Advances in Neural Information Processing Systems.* Morgan Kauffman, 1993.

[63] J. Takami and S. Sagayama. A successive state-splitting algorithm for efficient allophone modeling. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 573–576, 1992.

[64] F. Thollard, P. Dupont, and C. de la Higuera. Probabilistic DFA Inference using Kullback-Leibler Divergence and Minimality. In *Seventeenth International Conference on Machine Learning*, pages 975–982. Morgan Kauffman, 2000.

[65] M. Thomason and E. Granum. Dynamic programming inference of markov networks from finite sets of sample strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(4):491–501, 1986.

[66] W.-G. Tzeng. The equivalence and learning of probabilistic automata. In *30th Annual Symposium on Fundations of Computer Science*, pages 268–273, 1989.

[67] L.G. Valiant. A theory of the learnable. *Communications of the Association for Computing Machinery*, 27(11):1134–1142, 1984.

[68] R. Vasko, A. El-Jaroudi, and J. Boston. An algorithm to determine hidden markov model topology. In *International Conference on Acoustic, Speech and Signal Processing*, 1996.

[69] E. Vidal, P. Casacuberta, and P. García. Grammatical inference and applications to automatic speech recognition. In A.J. Rubio and J.M. López, editors, *NATO ASI, Speech Recognition and Coding, New Advances and Trends*, pages 174–191, 1995.

[70] A.J. Viterbi. Error bounds for convutional codes and asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.

[71] C. Wetherell. Probabilistic languages: A review and some open questions. *Computing Surveys*, 12(4):361–379, 1980.

[72] K. Yamanishi. A learning criterion for stochastic rules. *Machine Learning*, 9:165–203, 1992.

[73] Matthew Young-Lai and Frank Wm. Tompa. Stochastic grammatical inference of text database structure. *Machine Learning*, 40(2):111–137, 2000.