Machine Learning, 8, 151-166 (1992)
© 1992 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Learning Probabilistic Automata and Markov Chains via Queries

WEN-GUEY TZENG
Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794

Editor: Thomas G. Dietterich

Abstract. We investigate the problem of learning probabilistic automata and Markov chains via queries in the teacher-student learning model. Probabilistic automata and Markov chains are probabilistic extensions of finite state automata and have similar structures. We discuss some natural oracles associated with probabilistic automata and Markov chains. We present polynomial-time algorithms for learning probabilistic automata and Markov Chains using these oracles.

Keywords. Learning via oracles, probabilistic automaton, deterministic finite automaton, Markov chain.

1. Introduction

In this paper we use the teacher-student learning model (Angluin, 1987a). In the model a learner has access to oracles that provide correct information about the target concept. The goal of the learner is to learn the target concept exactly in polynomial time, with a polynomial number of queries. Informally, the teacher-student learning protocol can be described as follows. A teacher (oracle) has some target concept. A student (learning algorithm) tries to learn the concept by asking some questions (queries) about the concept. The student then constructs a hypothesis for the target concept and presents it to the teacher. The teacher tells the student whether the hypothesis is correct. If the hypothesis is incorrect, the teacher provides an instance (a counterexample) in which the hypothesis and the target concept disagree. The student uses the information provided by this counterexample to ask more questions and construct a new hypothesis. The student repeats this process until he/she presents a correct hypothesis to the teacher.

As an example, we describe a structure of the teacher-student protocol for learning deterministic finite automata (DFAs). Let A be a DFA and $L(A)$ be the language accepted by A. The target concept is $L(A)$. The student asks some *membership* questions (i.e., whether a string $x \in L(A)$). Then the student constructs a DFA A' and asks the teacher whether $L(A') = L(A)$ (an *equivalence* query). If the answer is "no," the teacher gives the student a counterexample, which is a string belonging to $L(A)$, but not to $L(A')$, or vice versa. Using information obtained from membership and equivalence queries, the student repeats the inference process again, until a correct DFA A'' is obtained, i.e., $L(A) = L(A'')$.

Intuitively, the learning ability of a student depends on the information the teacher provides. For example Angluin (1987a, 1989) showed that there is no polynomial-time algorithm for learning (i.e., exactly identifying) DFAs via membership queries alone or via equivalence queries alone. In contrast, she (Angluin, 1987b) showed that there is a polynomial-time algorithm that learns DFAs via a combination of membership and equivalence queries. A general problem in learning theory is to find, for each concept class, minimal sets of (natural) query types that enable the student to learn the class efficiently (in polynomial time by asking a polynomial number of queries). In this paper we present minimal sets of query types for learning probabilistic automata and Markov chains.

Probabilistic automata (PAs) and Markov chains (MCs) have been used to model some learning systems and pattern recognition problems (Bush & Mosteller, 1955; Fu, 1966). A PA is a finite state machine with probabilistic transitions among states. When a PA is in a state q and reads a symbol a, there is a fixed probability, for each state q', that the PA will move to state q'. These probabilities sum to 1. MCs are slightly different. They generate strings, rather than accepting them. When an MC is in state q, it has some probability p of generating the symbol "0" and probability $1 - p$ of generating the symbol "1" (we assume there are only two symbols). The next state it enters is determined by the symbol it generates. We treat PAs and MCs as functions that map strings into real numbers (the real numbers are accepting probabilities in the case of PAs and generating probabilities in the case of MCs). Both PAs and MCs are probabilistic generalizations of DFAs.

For a PA U, the *state distribution* (probability distribution of states) induced by an input string x is a vector indicating, for each state q in U, the probability that U enters state q after reading string x. We explore the SD oracle, which takes as input a string x and returns the state distribution induced by x for the target PA. Using a technique in linear algebra, we present a polynomial-time algorithm for learning PAs using the SD oracle. The same proof technique also yields a polynomial-time algorithm for the equivalence problem for PAs (Tzeng, 1990). We also show that this oracle is optimal for learning PAs in the following senses: (a) the consistency problem for PAs using information represented by the SD oracle is *NP*-complete and (b) it is provable that no polynomial-time algorithm can learn PAs using a weaker natural oracle.

When the SD oracle is restricted to DFAs, it takes as input a string x and returns the state q that the target DFA will enter after reading x. It is easy to see that we can learn DFAs using the SD oracle. However, we show that the corresponding consistency problem for DFAs is *NP*-complete. Each example used in the above consistency problem has the form $\langle x, q\rangle$, which says that the target DFA enters state q after reading string x. The information provided by examples of the form $\langle x, q\rangle$ is stronger than that which only indicates whether a string is accepted (Angluin, 1978; Gold, 1978).

For MCs, we discuss two types of oracles. The first one is similar to the membership oracle for DFAs. It takes as input a string x and outputs the probability that the string x is to be generated. The second one is similar to the equivalence oracel for DFAs. It takes as input an MC B, and if B is not equivalent to the target MC B^*, it outputs a string x that is generated with different probabilities by B and B^*. Our main results on learning MCs are as follows: (a) using the combination of these two oracles, we can learn MCs in polynomial time and (b) there is no polynomial-time algorithm for learning MCs using the first oracle alone or using the second oracle alone. These results are extensions of Angluin's (1987a, 1987b, 1989).

Rudich (1985) and DeSantis, et al. (1988) also discussed inference of MCs. They treated an MC as a device outputting an infinite sequence of symbols. Rudich showed how to infer the structure of the target MC with probability 1 in the limit by observing the output sequence. In contrast, we investigate the problem of exactly learning MCs in polynomial time using information about finite strings generated by the target MC.

In Valiant's PAC (probably approximately correct) learning model, we observe that examples are given probabilistically according to some probability distribution. Critical examples sometimes cannot be obtained through probabilistic sampling, because their share of probability measure in the sample space is negligibly small. Thus it is possible that some domains are learnable using oracles, but not learnable in Valiant's model (under the assumption that *NP*-hard problems cannot be solved in polynomial time by randomized algorithms, i.e., that $NP \neq RP$)[1]. Our results on learning PAs and MCs demonstrate a case where learning using oracles is more powerful than learning in Valiant's model. Other examples of concepts that are learnable using oracles, but not learnable in Valiant's model include, for example, read-once formulas (Angluin et al. 1989) and k-term DNF formulas (Angluin, 1987c; Kearns, et al. 1987).

2. Learning probabilistic automata

2.1. Definitions

A (row) vector is *stochastic* if all its entries are greater than or equal to 0 and sum to 1. A matrix is *stochastic* if all its row vectors are stochastic. let $\mathcal{M}(i, j)$ be the set of all $i \times j$ stochastic matrices. Let λ be the empty string and $|x|$ be the length of string x. Let α^T be the transpose of the vector α.

We first give definitions for deterministic finite automata and probabilistic automata in the following.

Definition 2.1 *A deterministic finite automaton (DFA) A is a 5-tuple $(Q, \Sigma, \delta, q_1, F)$, where Q is a finite set of states, Σ is an input alphabet, δ is a transition function from $Q \times \Sigma$ into Q, q_1 is the initial state, and $F \subseteq Q$ is a set of final states.*

The size of A is defined to be the number of states in Q and symbols in Σ. The transition function δ is extended to the domain $Q \times \Sigma^*$ in the standard way, i.e., for any $q \in Q$, $\sigma \in \Sigma$ and string x, $\delta(q, x\sigma) = \delta(\delta(q, x), \sigma)$ and $\delta(q, \lambda) = q$.

Definition 2.2 *Let A_1 and A_2 be two DFAs. Then A_1 and A_2 are said to be* equivalent *if $L(A_1) = L(A_2)$, where $L(A_1)$ and $L(A_2)$ are the languages accepted by A_1 and A_2 respectively.*

Definition 2.3 *A probabilistic automaton (PA) U is a 5-tuple (Q, Σ, M, ρ, F), where $Q = \{q_1, q_2, \ldots, q_n\}$ is a finite set of states, Σ is an input alphabet, M is a function from Σ into $\mathcal{M}(n, n)$, ρ is an n-dimensional stochastic row vector and $F \subseteq Q$ is a set of final states.*

The size of U is defined to be the number of states in Q and symbols in Σ. The vector ρ is called an *initial-state distribution* whose *ith* component is the probability of state q_i

being the initial state. Let $\sigma \in \Sigma$. The value $M(\sigma)[i, j]$ is the probability that U moves from state q_i to state q_j after reading σ. We extend the domain of function M from Σ to Σ^* in the standard way, i.e., $M(x\sigma) = M(x)M(\sigma)$. Let η_F be an n-dimensional row vector such that for $1 \leq i \leq n$,

$$\eta_F[i] = \begin{cases} 1 & \text{if } q_i \in F \\ 0 & \text{otherwise.} \end{cases}$$

Definition 2.4 *Let $U = (Q, \Sigma, M, \rho, F)$ be a PA. Then the* state distribution *$P_U(x)$ induced by string x is $\rho M(x)$, whose* ith *component is the probability that U with initial-state distribution ρ moves to state $q_i \in Q$ after reading x.*

The accepting probability *of x by U is $P_U(x)(\eta_F)^T$, which is the probability that U enters a final state when the input string is x.*

Definition 2.5 *Let $U_1 = (Q_1, \Sigma, M_1, \rho_1, F_1)$ and $U_2 = (Q_2, \Sigma, M_2, \rho_2, F_2)$ be two PAs. Then U_1* and *U_2 are said to be* equivalent *if for each string x, the accepting probability of x by U_1 is equal to the accepting probability of x by U_2, i.e., $\forall x \in \Sigma^*$, $P_{U_1}(x)(\eta_{F_1})^T = P_{U_2}(x)(\eta_{F_2})^T$.*

U_1 and U_2 are said to be state-distribution equivalent *if for each string x, the state distribution induced by x for U_1 is equal to the state distribution induced by x for U_2, i.e., $\forall x \in \Sigma^*$, $P_{U_1}(x) = P_{U_2}(x)$.*

We now give the definition for polynomial-time learning in the oracle learning model in the following.

Definition 2.6 *Let R be a class to be learned and O_R be an oracle for R. Then R is said to be* polynomially learnable using the oracle *O_R if there is a learning algorithm L and a two-variable polynomial p such that for every target $r \in R$ of size n, L runs in time $p(n, m)$ at any point and outputs a hypothesis that is equivalent to r, where m is the maximum length of data returned by O_R so far in the run.*

The above definition can be easily extended to the case where two or more oracles are used. The reason that the learning algorithm must run polynomially in every intermediate step is that otherwise the algorithm could delay outputting a correct hypothesis until after it found one (not necessarily in polynomial time). It could then ask the (equivalence) oracle about an hypothesis that differed from the target concept on very long strings only. The oracle would be forced to output a very long counterexample. Such an algorithm would run in "polynomial time" by using the size of such counterexamples as the complexity measure.

Since the computation of real numbers is a subtle issue, we assume that probabilities associated with automata are rational numbers and that each arithmetic operation of rational numbers can be done in constant time.

Without loss of generality, in the rest of this paper we assume that $\Sigma = \{0, 1\}$, unless stated otherwise.

2.2. Learnability of probabilistic automata

We first consider a natural oracle AP for PAs. For a PA $U = (Q, \Sigma, M, \rho, F)$, the AP oracle takes as input a string x and returns the accepting probability $P_U(x)(\eta_F)^T$ of x by U. This oracle is identical to the membership oracle if it is applied to DFAs. For a DFA A, the membership oracle returns "yes" (i.e., the accepting probability is 1) for a string x if and only if x is accepted by A. Since Angluin (1987a) has already shown that we cannot learn DFAs using the membership oracle alone, we cannot learn PAs using the AP oracle alone.

Theorem 2.7 ***(Angluin, 1987a)*** *DFAs are not polynomially learnable using the membership oracle only.*

Corollary 2.8 *PAs are not polynomially learnable using the* AP *oracle only.*

Therefore, we consider a stronger oracle SD for PAs. For a PA U, the SD oracle takes as input a string x and returns the state distribution $P_U(x)$ induced by x. To demonstrate that the information carried by this oracle is not too strong, we show that the consistency problem, based on the information carried by this oracle, for PAs is NP-complete.

First, we consider an analogous consistency problem for DFAs. For a DFA $A = (Q, \Sigma, \delta\ q_1, F)$, the SD oracle takes as input a string x and returns the state q that A will enter after reading x. In other words, the data about A presented in the corresponding consistency problem is given in the form $\langle x, q\rangle$, which says $\delta(q_1, x) = q$. The information carried by this type of data is stronger than that which only indicates whether a string is accepted (Angluin, 1978; Gold, 1978). It is easy to learn DFAs using the SD oracle alone. However, we show, in Theorem 2.10, that its corresponding consistency problem does not admit a polynomial-time algorithm if $P \neq NP$. Theorem 2.10 is quite strong in the sense that each state of the target DFA is reached (not just passed) by some string in the given data. The proof is a generalization of Angluin's (1978).

Theorem 2.9 *DFAs are polynomially learnable using the* SD *oracle.*

Proof: Since the proof is easy, we omit it here. □

Theorem 2.10 *The following problem is NP-complete: given a set T of data of the form $\langle x, q\rangle$, determine whether there is a DFA $A = (Q, \Sigma, \delta, q_1, \cdot)$ consistent with T, i.e., for each $\langle x, q\rangle$ in T, $\delta(q_1, x) = q$, such that $Q = \{q : \exists x, \langle x, q\rangle \in T\}$.*

Proof: It is easy to see that this problem is in NP. Then, we reduce the NP-complete problem SAT$'$ (Gold, 1978) with each clause containing positive literals only or negative literals only to our consistency problem. Let $V = \{v_1, v_2, \ldots, v_n\}$ be a set of variables and $C = \{c_1, c_2, \ldots, c_m\}$ be a set of clauses over V such that each c_i is either positive (containing positive literals only) or negative (containing negative literals only). For convenience, in the following we will use $\delta(q_1, x) = q$ to denote the example $\langle x, q\rangle$.

We first construct some tree-like automata A_V and A_{C_i}, $0 \le i \le n$, in Figure 1. The automaton A_V of height $\lceil \log n \rceil$ and state set Q_V has the root node q_v and leaf nodes q_{v_1}, $\ldots$, $q_{v_{n'}}$, where $n' = 2^{\lceil \log n \rceil}$. The leaf node q_{v_i} corresponds to the variable v_i of V. Let x_{v_i}, $1 \le i \le n'$, be the string of length $\lceil \log n \rceil$ such that $\delta(q_v, x_{v_i}) = \delta_{v_i}$. For each $0 \le i \le n$, the automaton A_{C_i} of height $\lceil \log m \rceil$ and state set Q_{C_i} has the root node q_{c_i} and leaf nodes $q_{c_i 1}, \ldots, q_{c_i m'}$, where $m' = 2^{\lceil \log m \rceil}$. The leaf node $q_{c_i j}$ corresponds to the clause c_j of C. Let x_{c_j}, $1 \le j \le m'$, be the string of length $\lceil \log m \rceil$ such that $\delta(q_{c_i}, x_{c_j}) = q_{c_i j}$. Let the state set Q with the initial state q_1 be $\{q_1, q_2, q_3, q_4, q_5, q_6\} \cup Q_V \cup (\cup_{i=0}^{n} Q_{C_i})$.

The transition (data) set T of our reduction is

$$T_1 \cup T_2 \cup T_3 \cup T_4 \cup T_5 \cup T_V \cup \left(\bigcup_{i=0}^{n} T_{C_i} \right),$$

where

$$T_1 : \delta(q_1, 0) = q_v,$$
$$\delta(q_1, 1) = q_{c_0},$$
$$\delta(q_1, 0x_{v_i}1) = \delta(q_{v_i}, 1) = q_{c_i},\ 1 \le i \le n,$$
$$T_2 : \delta(q_1, 0x_{v_i}1x_{c_j}0) = \begin{cases} q_2 & \text{if } v_i \text{ in } c_j \\ q_3 & \text{otherwise, } 1 \le i \le n,\ 1 \le j \le m, \end{cases}$$
$$T_3 : \delta(q_1, 1x_{c_j}01x_{c_j}0) = q_2,\ 1 \le j \le m,$$
$$T_4 : \delta(q_1, 1x_{c_j}00) = \begin{cases} q_4 & \text{if } c_j \text{ is positive} \\ q_5 & \text{otherwise, } 1 \le j \le m, \end{cases}$$
$$T_5 : \delta(q_i, y_i\, \sigma) = \delta(q_i\, \sigma) = q_6,\ \sigma \in \{0, 1\},$$
$$\text{where } \delta(q_1, y_i) = q_i,\ 2 \le i \le 6,$$
$$\delta(q_1, 0x_{v_i}\ 1x_{c_j}\ 0) = \delta(q_{c_i j}, 0) = q_6,\ 0 \le i \le n,\ m+1 \le j \le m',$$
$$\delta(q_1, 0x_{v_i}1x_{c_j}\ 1) = \delta(q_{c_i j}, 1) = q_6,\ 0 \le i \le n,\ 1 \le j \le m',$$
$$\delta(q_1, 0x_{v_i}\sigma) = \delta(q_{v_i}, \sigma) = q_6,\ n+1 \le i \le n',\ \sigma \in \{0, 1\},$$
$$T_V : \{\text{transitions of } A_V \text{ defined on } Q\},$$
$$T_{C_i} : \{\text{transitions of } A_{C_i} \text{ defined on } Q\},\ 0 \le i \le n.$$

The transition set T_5 collects all unrelated transitions by directing them to the state q_6. It is not hard to see that there is a DFA with state set Q on which the transitions in T_1, T_2, T_5, T_V, T_{C_i}, $0 \le i \le n$, can be defined. Thus the only transitions to be determined are those in

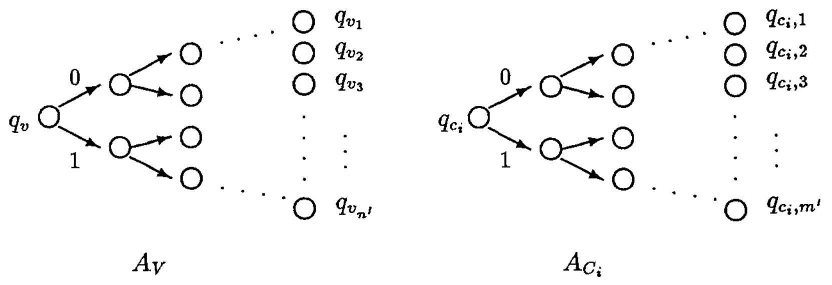

Figure 1. Automata A_V and A_{C_i}, $0 \le i \le n$.

$$R : \delta(q_1,\ 1x_{c_j}0),\ 1 \le j \le m,$$
$$\delta(q_1,\ 0x_{v_i}0) = \delta(q_{v_i},\ 0),\ 1 \le i \le n.$$

Now we prove that C is satisfiable if and only if there is a DFA with state set Q on which the transitions in T can be defined. For the forward direction, assume that τ is a truth assignment for V that satisfies C. We define a DFA $A = (Q, \Sigma, \delta, q_1, \cdot)$ such that, in addition to the transitions in T, the (undetermined) transitions in R are defined as

$$\delta(q_1,\ 1x_{c_j}0) = q_{v_k},\ 1 \le j \le m,$$

where the clause c_j is satisfied by $\tau(v_k)$, and

$$\delta(q_1,\ 0x_{v_i}0) = \delta(q_{v_i},0) = \begin{cases} q_4 & \text{if } \tau(v_i) = \textit{true} \\ q_5 & \text{if } \tau(v_i) = \textit{false},\ 1 \le i \le n. \end{cases}$$

We can check that all transitions in T, in particular, those in T_3 and T_4, are legal for A.

For the backward direction, let $A = (Q, \Sigma, \delta, q_1, \cdot)$ be a DFA on which transitions in T are defined. Then, transitions in T_2 and T_3 guarantee that for every $1 \le j \le m$, $\delta(q_1, 1x_{c_j}0) = q_{v_k}$ for some $1 \le k \le n$ and variable v_k is in the clause c_j. Furthermore transitions in T_4 guarantee that $\delta(q_1, 1x_{c_j}00) = \delta(q_{v_k}, 0) = q_4$ or q_5. So, we define the following truth assignment τ for C:

$$\tau(v_k) = \begin{cases} \textit{true} & \text{if } \delta(q_1,\ 1x_{c_j}00) = \delta(q_{v_k},\ 0) = q_4 \\ \textit{false} & \text{if } \delta(q_1,\ x_{c_j}00) = \delta(q_{v_k},\ 0) = q_5 \\ \textit{arbitrary} & \text{otherwise,} \end{cases}$$

such that a clause c_j is satisfied by $\tau(v_k)$ if $\delta(q_1,\ 1x_{c_j}0) = q_{v_k}$. □

The above theorem holds for the case of PAs.

Theorem 2.11 *The following problem is NP-complete: given a set T' of data of the form $\langle x, \vec{p}\rangle$, determine whether there is a PA U consistent with T', i.e., for each $\langle x, \vec{p}\rangle$ in T', $P_U(x) = \vec{p}$.*[2]

Proof: We use the same construction (reduction) as in the previous proof, except that we use state distributions to represent transitions. Let $\vec{p}[q]$ be the coordinate that corresponds to the state q in Q of the previous proof. Then the set T' of state distributions is defined as

$$\{\langle x, \vec{p}\rangle : \exists q, \langle x, q\rangle \in T, \vec{p}[q] = 1, \vec{p}[q'] = 0 \text{ if } q' \neq q\}.$$

The same argument as in the previous proof proves that if C is satisfiable then there exists a PA U with state set Q on which state distributions in T' can be defined. Proving the converse of this statement is less straightforward.

In the previous proof, we assigned a truth value to a variable v_k depending on whether $\delta(q_1, 1x_{c_j}00) = \delta(q_{v_k}, 0)$ was equal to q_4 or q_5. In the case of PAs, the state distributions in R' (which corresponds to R in the previous proof) may reach different states with different probabilities. This may appear to cause a problem in using the above method to assign truth values to the variables. We will show, however, that there is no such problem. The state distributions in T_2' and T_3' (which corresponds to T_2 and T_3 in the previous proof respectively) guarantee that U, on input $1x_{c_j}0$, will reach some state q_{v_k} in $\{q_{v_{j_1}}, \ldots, q_{v_{j_r}}\}$ with probability 1, where $v_{j_1}, \ldots, v_{j_r}$ are the variables appearing in the clause c_j. Otherwise the probability that U, on input $1x_{c_j}01x_{c_j}0$, reaches q_2 is not 1, violating state distributions in T_3'. But when U, on input $1x_{c_j}0$, reaches q_{v_k} with probability 1, it, on input $1x_{c_j}00$, either goes to q_4 with probability 1 or to q_5 with probability 1. Otherwise it violates state distributions in T_4' (which corresponds to T_4 in the previous proof). Therefore, we may use the method of the previous proof to assign truth values to the variables. □

We can relax the requirement of the state set being $Q = \{q : \exists x, \langle x, q\rangle \in T\}$ in Theorem 2.10 to get a non-approximation result for the case where the given data has the form $\langle x, q\rangle$.

Theorem 2.12 *The following problem is NP-hard for any fixed polynomial p: given a set T of data of the form $\langle x, q\rangle$, find a DFA consistent with T and of number of states at most $p(n^*)$, where n^* is the minimum number of states of any consistent DFA.*

Proof: (Sketch) The proof is a simple reduction from the non-approximation result of DFAs (Pitt & Warmuth, 1989). In Pitt and Warmuth (1989), an example of the given set T' has the form $\langle x, \varphi\rangle$ such that string x is accepted if and only if φ is $+$. Let @ be a new alphabet symbol and q_+, q_-, q_d be three new states. The set T of our reduction is

$$\begin{aligned}
& \{\langle x@, q_+\rangle : \langle x, +\rangle \in T'\}\\
\cup\ & \{\langle x@, q_-\rangle : \langle x, -\rangle \in T'\}\\
\cup\ & \{\langle @, q_+\rangle, \langle @0, q_-\rangle, \langle @1, q_-\rangle, \langle @00, q_+\rangle, @01, q_+\rangle\}\\
\cup\ & \{\langle x, q_d\rangle : x = @@, @0, @1, @0@, @00, @01, @@@, @@0, @@1\}.
\end{aligned}$$

Now we can see that there is an n-state DFA consistent with T' if and only if there is an $(n + 3)$-state DFA consistent with T. Thus the non-approximation result of Pitt and Warmuth (1989) also holds for the new form of data. □

2.3. *Learning probabilistic automata using the* **SD** *oracle*

In this section we present a polynomial-time algorithm for learning PAs using the SD oracle.

Theorem 2.13 *PAs are polynomially learnable using the* SD *oracle even if the output hypothesis of the learning algorithm must be state-distribution equivalent to the target PA.*

Corollary 2.14 *PAs are polynomially learnable using the* SD *oracle.*

The rest of this section is devoted to prove Theorem 2.13.

2.3.1. *Overview*

We first review the result on the equivalence problem for PAs in Tzeng (1990), because the ideas and techniques are crucial to our learning algorithm.

Let $U_1 = (Q_1, \Sigma, M_1, \rho_1, F_1)$ and $U_2 = (Q_2, \Sigma, M_2, \rho_2, F_2)$ be two PAs of n_1 and n_2 states respectively. We define

$$M_{U_1 \oplus U_2}(x) = \begin{bmatrix} M_1(x) & \mathbf{0}_{n_1 \times n_2} \\ \mathbf{0}_{n_2 \times n_1} & M_2(x) \end{bmatrix},$$

where $\mathbf{0}_{n \times m}$ is the $(n \times m)$-dimensional zero matrix. Let $P_{U_1 \oplus U_2}(x) = [\rho_1, \rho_2] M_{U_1 \oplus U_2}(x)$ and $H = \{P_{U_1 \oplus U_2}(x) : x \in \Sigma^*\}$. Then U_1 and U_2 are equivalent if and only if $\forall x \in \Sigma^*$, $P_{U_1 \oplus U_2}(x)[\eta_{F_1}, -\eta_{F_2}]^T = 0$, i.e., *span*$(H)$ is a null space for $[\eta_{F_1}, -\eta_{F_2}]^T$, where *span* is the function that maps a set of vectors to the vector space generated by the vectors in the set. The key idea of designing a polynomial-time algorithm for the equivalence problem for PAs is to find a basis V for *span*(H) in polynomial time. Then U_1 and U_2 are equivalent if and only if $\forall \vec{v} \in V$, $\vec{v}[\eta_{F_1}, -\eta_{F_2}]^T = 0$.

We define a binary tree T as follows. The root of tree T is *node*(λ) and each *node*(x) (x is a string) has two children *node*$(x0)$ and *node*$(x1)$. Let $P_{U_1 \oplus U_2}(x)$ be the $(n_1 + n_2)$-dimensional vector associated with *node*(x). For *node*$(x\sigma)$, $\sigma \in \Sigma$, its associated vector $P_{U_1 \oplus U_2}(x\sigma)$ can be calculated by multiplying its father's associated vector $P_{U_1 \oplus U_2}(x)$ by $M_{U_1 \oplus U_2}(\sigma)$.

The method we use to determine whether the two PAs are equivalent is to prune the tree T. Initially, we set V to be the empty set. Then, we visit tree T in breadth-first order. At each node, we verify whether the associated vector of the currently visited node is linearly independent of V. If it is so, we add the vector to V. Otherwise we prune the sub-tree rooted at the current node. We stop traversing tree T when every node in T is either visited or pruned. The vectors in the resulting set V form a basis for *span*(H). Actually the order of traversal can be arbitrary. Different orders of traversal result in different bases for *span*(H).

Lemma 2.15 (*Tzeng, 1990*) *There is a polynomial-time algorithm that takes as input two PAs U_1 and U_2, and determines whether U_1 and U_2 are equivalent.*

2.3.2. *The algorithm*

Let the target PA be $U^* = (Q, \Sigma, M^*, \rho^*, F)$, where Q and F are known. Our algorithm for learning PAs using the **SD** oracle appears in Table 1. By the definition of the **SD** oracle, we can see that $\rho^* = \vec{p}_\lambda = \mathsf{SD}(\lambda)$. In the first part of the algorithm (lines 1-9), we find a basis V for the vector space $span(\{P_{U^*}(x) : x \in \Sigma^*\})$ by the same techniques we used to solve the equivalence problem for PAs. Then, in the second part of the algorithm (lines 10-14), we define a system of linear equations from V and solve it to get an *arbitrary* solution for the transition matrices $M(\sigma)$, $\sigma \in \Sigma$. We show, in the following subsection, that the inferred PA $U = (Q, \Sigma, M, \vec{p}_\lambda, F)$ is state-distribution equivalent to the target U^*.

Table 1. Learning algorithm for probabilistic automata.

```
1.  $\vec{p}_\lambda \leftarrow$ SD($\lambda$);
2.  Set $V$ to be the empty set;
3.  $W \leftarrow \{node(\lambda)\}$;
4.  while $W$ is not empty do
5.  begin Take an element $node(x)$ from $W$;
6.      $\vec{p}_x \leftarrow$ SD($x$);
7.      if $\vec{p}_x \notin span(V)$
8.      then begin Add $node(x0)$ and $node(x1)$ to $W$;
9.                 $V \leftarrow V \cup \{\vec{p}_x\}$ end;
    end;
10. Let $M(0) = [x_{ij}]$ and $M(1) = [y_{ij}]$, $1 \le i, j \le n$;
11. Define a linear system:
        for $\vec{p}_x \in V$ and $\sigma \in \{0, 1\}$, $\vec{p}_x M(\sigma) = \vec{p}_{x\sigma} =$ SD($x\sigma$),
        for $1 \le i \le n$, $\Sigma_{j=1}^{n} x_{ij} = 1$, $\Sigma_{j=1}^{n} y_{ij} = 1$,
        for $1 \le i, j \le n$, $x_{ij} \ge 0$, $y_{ij} \ge 0$;
12. Find a suitable solution for $x_{ij}$'s and $y_{ij}$'s;
13. if there is a solution then return ($U = (Q, \{0, 1\}, M, \vec{p}_\lambda, F)$)
14. else return (not exist);
```

2.3.3. *Correctness*

We prove that the PA U returned by the above algorithm is state-distribution equivalent to the target PA U^*. Note that the transition function M of U may not be identical to M^* of U^*, but their behaviors with respect to the state distributions for strings are the same.

Lemma 2.16 *For all strings $x \in \Sigma^*$, $P_U(x) = P_{U^*}(x)$.*

Proof: Let $V = \{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_r\}$ be the set of vectors obtained by the above algorithm in $node(x_1)$, $node(x_2)$, $\ldots$, $node(x_r)$ and $N = \{node(x_i) : 1 \le i \le r\}$. Let T_N be the tree formed by the nodes in N. Let $N_0 = N$ and $N_i = \{node(xy) : node(x)$ is a leaf of T_N and $|y| = i\}$, which is the set of nodes of distance i from T_N. Then $\cup_{i=0}^{\infty}\{N_i\} = \{node(x) : x \in \Sigma^*\}$. Note that the vectors in V form a basis for $span(\{P_{U^*}(x) : x \in \Sigma^*\})$. We prove this lemma by induction on i.

Base. For all $node(x) \in N_0 \cup N_1$, $P_U(x) = P_{U^*}(x)$ follows from the algorithm.

Hypothesis. For all $node(x) \in N_i$, $P_U(x) = P_{U^*}(x)$ holds for i.

Step. For $node(x\sigma) \in N_{i+1}$, where $node(x) \in N_i$ and $\sigma \in \Sigma$,

$$\begin{aligned} P_U(x\sigma) &= P_U(x)M(\sigma) = P_{U^*}(x)M(\sigma) = \left(\sum_{i=1}^{r} m_i\vec{v}_i\right) M(\sigma) \\ &= \sum_{i=1}^{r} m_i(\vec{v}_iM(\sigma)) = \sum_{i=1}^{r} m_i(\vec{v}_iM^*(\sigma)) = \left(\sum_{i=1}^{r} m_i\vec{v}_i\right) M^*(\sigma) \\ &= P_{U^*}(x)M^*(\sigma) = P_{U^*}(x\sigma) \end{aligned}$$

□

2.3.4. *Complexity*

The first part of the algorithm is polynomial-time solvable by Lemma 2.15. The second part defines and solves a system of linear equations under the positive variable constraint. This is a typical linear programming problem. We only need a feasible solution which is not necessarily optimal. The problem of linear programming has been known to be polynomial-time solvable (Khachiyan, 1979; Karmarkar, 1984). So, our learning algorithm runs polynomially in n, where n is the number of states of the target PA U^*.

2.3.5. *Remark*

The requirement that, for each state q_i and each alphabet symbol σ, the outgoing probabilities of state q_i on σ sum to 1 (i.e., $\Sigma_{j=1}^{n} M(\sigma)[i, j] = 1$) is not essential to our results.

Another natural definition for PA is that, for each state q_i, the outgoing probabilities of state q_i on all alphabet symbols sum to 1 (i.e., $\Sigma_{\sigma\in\Sigma}\ \Sigma_{j=1}^{n}\ M(\sigma)[i, j] = 1$). For learning this type of PA using the (modified) SD oracle, we only need replace "for $1 \leq i \leq n$, $\Sigma_{j=1}^{n} x_{i,j} + \Sigma_{j=1}^{n} y_{i,j} = 1$" with "for $1 \leq i \leq n$, $\Sigma_{j=1}^{n} x_{i,j} = 1$, $\Sigma_{j=1}^{n} y_{i,j} = 1$" in line 11 of the algorithm.

3. Learning Markov chains

In this section we discuss the learnability of Markov chains via queries. Let [0, 1] be the set of real numbers between 0 and 1. Let $\pi_i(\alpha)$ be the ith component of the tuple α.

Definition 3.1 *A Markov chain (MC) B is a 4-tuple $(Q, \Sigma, \mathcal{P}, q_1)$, where Q is a finite set of states, Σ is a finite alphabet, $\mathcal{P}$ is a function from $Q \times \Sigma$ into $Q \times [0, 1]$ such that for every $q \in Q$ $\Sigma_{\sigma\in\Sigma}\pi_2(\mathcal{P}(q, \sigma)) = 1$, and q_1 is the initial state.*

The size of B is defined to be the number of states in Q and symbols in Σ. We treat MCs as probabilistic symbol-generating automata. When B is in state q, it has probability $\pi_2(\mathcal{P}(q, 0))$ of generating symbol "0" and probability $1 - \pi_2(\mathcal{P}(q, 0))$ of generating symbol "1" (for the case that $\Sigma = \{0, 1\}$). It then enters the next state according to the symbol it generates. The *generating probability* of x by B is the product of the probabilities associated with the transitions passed by B when x is generated. This probability is relative to strings of length $|x|$. Let $g_B(x)$ be the function that maps a string x to its generating probability by B.

Definition 3.2 *Let B_1 and B_2 be two MCs. Then B_1 and B_2 are said to be* equivalent *if for each string x, the generating probability of x by B_1 is equal to the generating probability of x by B_2 i.e., $\forall x \in \Sigma^*$, $g_{B_1}(x) = g_{B_2}(x)$.*

We consider two oracles GP and EQ for MCs. For an MC B, the GP oracle takes as input a string x and returns the generating probability $g_M(x)$ of x. The EQ oracle takes as input an MC B', and if B and B' are equivalent, it returns "yes." Otherwise it returns a string x and its generating probability $g_B(x)$ by B such that $g_B(x) \neq g_{B'}(x)$. These two oracles are similar to and a little stronger than the membership and equivalence oracles for DFAs respectively.

Our results about learning MCs using the GP and EQ oracles are similar to Angluin's results about learning DFAs using the membership and equivalence oracles (Angluin, 1987a, 1987b, 1989). The proof techniques we use are also similar to hers. We state our results about learning MCs using oracles in the following three theorems and sketch how to modify Angluin's proofs for them.

Theorem 3.3 *MCs are not polynomially learnable using the* GP *oracle only.*

Proof: (Sketch) Assume that there is a polynomial-time algorithm Π that exactly identifies MCs using the GP oracle. Let $p \neq 1/2$ be a number between 0 and 1. We prove the theorem by an adversary argument. We first construct a set S of MCs $B = (\{q_1, \ldots, q_n\}, \{0, 1\}, \mathcal{P}, q_1)$, where

$$\begin{aligned}
\mathcal{P}(q_i, 0) &\in \{(q_{i+1}, 1/2), (q_1, 1/2)\}, \ 1 \leq i \leq n-1, \\
\mathcal{P}(q_i, 1) &= \begin{cases} (q_{i+1}, 1/2) & \text{if } \mathcal{P}(q_i, 0) = (q_1, 1/2) \\ (q_1, 1/2) & \text{if } \mathcal{P}(q_i, 0) = (q_{i+1}, 1/2), \ 1 \leq i \leq n-1, \end{cases} \\
\mathcal{P}(q_n, 0) &= (q_n, p).
\end{aligned}$$

There are 2^{n-1} MCs in the set S. Any two MCs in S are not equivalent. When Π queries GP on string x of length m, if there is at least one MC in S that generates string x with probability $1/2^m$ then the oracle returns $1/2^m$. There are at most $m - n + 2$ MCs in S that do not generate string x with probability $1/2^m$ if $m \geq n - 1$. So, when algorithm Π makes a query, only a few (polynomial) number of MCs in S need be removed to maintain consistency. Therefore, the number of MCs in S (which initially contains 2^{n-1} MCs) cannot be reduced to 1 by only a polynomial number of GP queries. Thus the polynomial-time learning algorithm Π does not exist. □

Theorem 3.4 *MCs are not polynomially learnable using the* EQ *oracle only.*

Proof: (Sketch) The construction of a set of MCs to be used for the adversary argument is basically the one used in Angluin (1989). We refer readers to her paper and only give a sketch of modifying her construction. Let @ be a new alphabet symbol and q_r, q_f be two new distinguished states. Let $p_1 \neq 1/3$ and $p_2 \neq 1/3$ be two different numbers between 0 and 1. For each DFA $A = (Q, \Sigma, \delta, q_1, F)$ in Angluin's construction, we construct a corresponding MC $B = (Q \cup \{q_r, q_f\}, \Sigma \cup \{@\}, \mathcal{P}, q_1)$, where

$$\begin{aligned}
\mathcal{P}(q, \sigma) &= (\delta(q, \sigma), 1/3) \text{ for } q \in Q \text{ and } \sigma \in \{0, 1\}, \\
\mathcal{P}(q, @) &= (q_r, 1/3) \text{ for } q \in Q - F, \\
\mathcal{P}(q, @) &= (q_f, 1/3) \text{ for } q \in F, \\
\mathcal{P}(q_r, 0) &= (q_r, p_1), \ \mathcal{P}(q_r, 1) = (q_r, 1 - p_1), \ \mathcal{P}(q_r, @) = (q_r, 0), \\
\mathcal{P}(q_f, 0) &= (q_f, p_2), \ \mathcal{P}(q_f, 1) = (q_f, 1 - p_2), \ \mathcal{P}(q_f, @) = (q_f, 0).
\end{aligned}$$

Any two MCs in the set we constructed are not equivalent. For each counterexample $\langle x, \varphi \rangle$ returned by the equivalence oracle of DFAs, the corresponding counterexample returned by the EQ oracle would be string $x@0$ with generating probability $p_1/3^{m+1}$ if $\varphi = -$, and $x@0$ with generating probability $p_2/3^{m+1}$ if $\varphi = +$, where $|x| = m$. Then, the adversary argument in Angluin (1989) can be applied to the case of MCs in the same way. □

Theorem 3.5 *MCs are polynomially learnable using the* GP *and* EQ *oracles.*

Proof: (Sketch) First, we extend Angluin's (1987b) learning algorithm for DFAs using the membership and equivalence oracles to learn *multi-class* DFAs using the (modified) membership and (modified) equivalence oracles. A multi-class DFA is a DFA with each state being classified into a class. Thus a (standard) DFA is a two-class DFA with the "acceptance" and "nonacceptance" classes. Two multi-class DFAs A_1 and A_2 are equivalent if and only if for each string x, A_1 and A_2, both on input x, enter states in the same class. For a multi-class DFA A, the modified membership oracle takes as input a string x and returns the class of the state that A, on input x, enters. The modified equivalence oracle takes as input a multi-class DFA A', and if A and A' are equivalent, it returns "yes." Otherwise it returns a string x and the class r of the state that A, on input x, enters such that A', on input x, does not enter a state of class r.

Angluin's algorithm is basically a refinement algorithm. In the beginning, the algorithm partitions the states of the target DFA into two (acceptance and nonacceptance) classes. Then using the counterexamples provided by the equivalence oracle and querying the membership oracle, it further separates states in the same class until no further separation is possible. The learning algorithm, using the (modified) membership and (modified) equivalence oracles, for multi-class DFAs is an easy extension of Angluin's by first grouping states in the same class together and then refining them by the information provided by oracles. We refer to Angluin's paper (1987b) for the learning algorithm for DFAs using the membership and equivalence oracles.

For MCs, we identify a class by a number. Our basic observation is that we can classify states of MCs into classes by their transition probabilities. For an MC $B = (Q, \Sigma, \mathcal{P}, q_1)$, state $q \in Q$ is in the class $\pi_2(\mathcal{P}(q, 0))$ (for the case of the two-letter alphabet). Two states q and q' are in the same class if and only if $\pi_2(\mathcal{P}(q, 0)) = \pi_2(\mathcal{P}(q', 0))$. These classes defined by numbers are analogous to the classes of multi-class DFAs. Like querying the (modified) membership oracle about the class of the state that the target multi-class DFA, on input x, enters, we can query the GP oracle to obtain the class of the state that the target MC enters when a string x is generated. When string x is generated by B, B enters a state in the class $g_B(x0)/g_B(x)$. The EQ oracle for MCs is analogous to the (modified) equivalence oracle for multi-class DFAs. Thus the learning algorithm, using the (modified) membership and (modified) equivalence oracles, for multi-class DFAs can be applied to learn MCs using the GP and EQ oracles directly. □

4. Conclusion

By studying the basic properties of PAs and MCs, we explored the learnability of PAs and MCs by examples and by using oracles. We investigated natural oracles associated with PAs and MCs. For PAs, we presented a polynomial-time learning algorithm using the SD oracle. We showed that PAs are not learnable using the weaker AP oracle. We also showed that the consistency problem from the information carried by the SD oracle for PAs (or, DFAs) is *NP*-complete. For MCs, we presented a polynomial-time algorithm using both the GP and EQ oracles. We showed that if we drop either of the GP and EQ oracles then no polynomial-time algorithm exits.

Acknowledgments

The author would like to thank Professor Ker-I Ko for his support, encouragement, and many helpful discussions. The author also thanks the referees for their valuable comments. Part of this paper appeared in the extended abstract "The equivalence and learning of probabilistic automata." Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science (1989). Los Alamitos, CA: IEEE Press. The work of the author was supported in part by NSF grant CCR-8801575.

Notes

1. There are several variations of the definition for PAC-learnability that have been used in the literature (Valiant, 1984; Haussler, et al., 1988; Pitt & Warmuth, 1990). Here we adopt the definition that requires the output hypothesis to be in the form of the original domain. A more general version of PAC-learnability does not require any fixed representation domain for the output hypothesis. This is sometimes called PAC-predictability. It has been pointed out that *NP*-completeness of the consistency problem for a domain does not imply that the domain is not PAC-predictable under the assumption $NP \neq RP$ (Pitt & Warmuth, 1989; Pitt, 1990).
2. Since data of the form $\langle x, \vec{p} \rangle$ fixes the number of states of U to be the dimension of $\vec{p}$, the state set is not an issue for the case of PAs.

References

Angluin, D. (1978). On the complexity of minimum inference of regular sets. *Information and Control, 39,* 337–350.

Angluin, D. (1981). A note on the number of queries needed to identify regular languages. *Information and Control, 51,* 76–87.

Angluin, D. (1987a). Queries and concept learning. *Machine Learning, 2,* 319–342.

Angluin, D. (1987b). Learning regular sets from queries and counterexamples. *Information and Control, 75,* 87–106.

Angluin, D. (1987c). *Learning k-term DNF formulas using queries and counterexamples* (Technical Report YALEU/DCS/RR-559). New Haven, CT: Yale University, Computer Science Department.

Angluin, D. (1988). *Negative results for equivalence queries* (Technical Report YALEU/DCS/RR-648). New Haven, CT: Yale University, Computer Science Department.

Angluin, D. (1989). Equivalence queries and approximate fingerprints. *Proceedings of the Second Workshop on Computational Learning Theory* (pp. 134–145). San Mateo, CA: Morgan Kaufmann.

Angluin, D., Hellerstein, L., & Karpinski, M. (1989). *Learning read-once formulas with queries* (Technical Report UCB/CSD 89-528). Berkeley, CA: University of California-Berkeley, Computer Science Division (EECS).

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M.K. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery, 36,* 929-965.

Board, R., & Pitt, L. (1990). On the necessity of Occam algorithms. *Proceedings of the Twenty-Second Annual Symposium on Theory of Computing* (pp. 54–63). New York: NY: ACM Press.

Bush, R.R., & Mosteller, F. (1955). *Stochastic models for learning.* New York, NY: Wiley.

DeSantis, A., Markowsky, G., Wegman, M.N. (1988). Learning probabilistic prediction functions, *Proceedings of the Twenty-Ninth Annual Symposium on Foundations of Computer Science* (pp. 110–119). Los Alamitos, CA: IEEE Press.

Fu, K.S. (1966). Stochastic automata as models of learning systems. *Proceedings of Symposium on Computer and Information Science.* Columbus, OH: Purdue University.

Gold, E.M. (1978). Complexity of automaton identification from given data. *Information and Control, 37,* 302–320.

Haussler, D., Kearns, M., Littlestone, M., & Warmuth, M.K. (1988). Equivalence of models for polynomial learnability. *Proceedings of the First Workshop on Computational Learning Theory* (pp. 42–55). San mateo, CA: Morgan Kaufmann.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica, 4,* 373–395.

Kearns, M., Li, M., Pitt, L., & Valiant, L.G. (1987). On the learnability of boolean formulae. *Proceedings of the Nineteenth Annual Symposium on Theory of Computing* (pp. 285–295). New York, NY: ACM Press.

Kearns, M., & Valiant, L.G. (1989). Cryptographic limitations on learning Boolean formulae and finite automata. *Proceedings of the Twenty-First Annual Symposium on Theory of Computing* (pp. 433–444). New York, NY: ACM Press.

Khachiyan, L.G. (1979). A polynomial algorithm in linear programming (*in Russian*). Translated in *Soviet Mathematics Doklady, 20,* 191–194.

Paz, A. (1971) *Introduction to probabilistic automata.* New York, NY: Academic Press.

Pitt, L. (1989). *Inductive inference, DFAs, and computational complexity* (Technical Report UIUCDCS-R-89-1530). Champaign, IL: Univeristy of Illinois at Urbana-Champaign, Department of Computer Science.

Pitt, L., & Warmuth, M.K. (1989). The minimum consistent DFA problem cannot be approximated within any polynomial. *Proceedings of the Twenty-First Annual Symposium on theory of Computing* (pp. 421–432). New York, NY: ACM Press.

Pitt, L., & Warmuth, M.K. (1990). Prediction-preserving reducibility. *Journal of Computer and System Sciences, 43,* 430–467.

Rabin, M.O. (1963). Probabilistic automata. *Information and Control, 6,* 230–245.

Rudich, S. (1985). Inferring the structure of a Markov chain from its output. *Proceedings of the Twenty-Sixth Annual Symposium on Foundations of Computer Science* (1989) (pp. 321–326). Los Alamitos, CA: IEEE Press.

Tzeng, W. (1990). A polynomial-time algorithm for the equivalence of probabilistic automata. Submitted for publication.

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