On the Learnability of Shuffle Ideals

Journal of Machine Learning Research 14 (2013) 1513-1531 Submitted 10/12; Published 6/13

On the Learnability of Shuffle Ideals

Dana Angluin DANA.ANGLUIN@YALE.EDU
James Aspnes JAMES.ASPNES@YALE.EDU
Department of Computer Science
Yale University
New Haven, CT 06520 USA

Sarah Eisenstat SEISENST@MIT.EDU
CSAIL, MIT
Cambridge, MA 02139 USA

Aryeh Kontorovich KARYEH@CS.BGU.AC.IL
Department of Computer Science
Ben-Gurion University of the Negev
Beer Sheva, Israel 84105

Editor: Mehryar Mohri

Abstract

PAC learning of unrestricted regular languages is long known to be a difficult problem. The class of shuffle ideals is a very restricted subclass of regular languages, where the shuffle ideal generated by a string u is the collection of all strings containing u as a subsequence. This fundamental language family is of theoretical interest in its own right and provides the building blocks for other important language families. Despite its apparent simplicity, the class of shuffle ideals appears quite difficult to learn. In particular, just as for unrestricted regular languages, the class is not properly PAC learnable in polynomial time if RP $\neq$ NP, and PAC learning the class improperly in polynomial time would imply polynomial time algorithms for certain fundamental problems in cryptography. In the positive direction, we give an efficient algorithm for properly learning shuffle ideals in the statistical query (and therefore also PAC) model under the uniform distribution.

Keywords: PAC learning, statistical queries, regular languages, deterministic finite automata, shuffle ideals, subsequences

1. Introduction

Inferring regular languages from examples is a classic problem in learning theory. A brief sampling of areas where various automata show up as the underlying formalism include natural language processing (speech recognition, morphological analysis), computational linguistics, robotics and control systems, computational biology (phylogeny, structural pattern recognition), data mining, time series and music (Koskenniemi, 1983; de la Higuera, 2005; Mohri, 1996; Mohri et al., 2002; Mohri, 1997; Mohri et al., 2010; Rambow et al., 2002; Sproat et al., 1996). Thus, developing efficient formal language learning techniques and understanding their limitations is of a broad and direct relevance in the digital realm.

Perhaps the currently most widely studied theoretical model of learning is Valiant's PAC model, which allows for a clean, elegant theory while retaining some measure of empirical plausibility (Valiant, 1984). Since PAC learnability is characterized by finite VC-dimension and the concept

class of n-state deterministic finite state automata (DFA) has VC-dimension $\Theta(n \log n)$ (Ishigami and Tani, 1997), the PAC learning problem is solved, in an information theoretic sense, by constructing a DFA on n states consistent with a given labeled sample. Unfortunately, as shown in the works of Angluin (1978), Gold (1978) and Pitt and Warmuth (1993) under standard complexity assumptions, finding small consistent automata is a computationally intractable task. Furthermore, attempts to circumvent the combinatorial search over automata by learning with a different representation class are thwarted by cryptographic hardness results. The papers of Pitt and Warmuth (1990) and Kearns and Valiant (1994) prove the existence of small automata and "hard" distributions over $\{0,1\}^n$ so that any efficient learning algorithm that achieves a polynomial advantage over random guessing will break various cryptographic hardness assumptions.

In a modified model of PAC, and with additional structural assumptions, a class of probabilistic finite state automata was shown by Clark and Thollard (2004) and Palmer and Goldberg (2007) to be learnable. If the target automaton and sampling distribution are assumed to be "simple", efficient probably exact learning is possible (Parekh and Honavar, 2001). When the learner is allowed to make membership queries, it follows by the results of Angluin (1987) that DFAs are learnable in this augmented PAC model.

The prevailing paradigm in regular language learning has been to make structural regularity assumptions about the family of languages and/or the sampling distribution in question and to employ a state merging heuristic. Indeed, over the years a number of clever and sophisticated combinatorial approaches have been proposed for learning DFAs. Typically, an initial automaton or prefix tree consistent with the sample is first created. Then, starting with the trivial partition with one state per equivalence class, classes are merged while preserving an invariant congruence property. The automaton learned is obtained by merging states according to the resulting classes. Thus, the choice of the congruence determines the algorithm and generalization bounds are obtained from the structural regularity assumptions. This rough summary broadly characterizes the techniques of Angluin (1982), Oncina and García (1992), Ron et al. (1998), Clark and Thollard (2004), Parekh and Honavar (2001) and Palmer and Goldberg (2007), and until recently this appears to have been the only general purpose technique available for learning finite automata.

More recently, Kontorovich et al. (2006), Cortes et al. (2007) and Kontorovich et al. (2008) proposed a substantial departure from the state merging paradigm. Their approach was to embed a specific family of regular languages (the piecewise-testable ones) in a Hilbert space via a kernel and to identify languages with hyperplanes. A unifying feature of this methodology is that rather than building an automaton, the learning algorithm outputs a classifier defined as a weighted sum of simple automata. In subsequent work by Kontorovich and Nadler (2009) this approach was extended to learning general discrete concepts. These results, however, provided only margin based generalization guarantees, which are weaker than true PAC bounds.

A promising research direction is to investigate the question of efficient PAC learnability for restricted subclasses of the regular sets. One approach is to take existing efficient PAC algorithms in other domains, for example, for classes of propositional formulas over the boolean cube $\{0,1\}^n$, or classes of geometric concepts such as axis-aligned boxes in $\mathbb{R}^n$, discretize the representation if necessary, and consider the resulting sets of strings to be formal languages. If the languages have finite cardinality, they are trivially regular, although they may or may not have succinct deterministic finite state acceptors.

Another approach is to consider classes of regular languages defined by structural restrictions on the automata or grammars that accept or generate them. Ergün et al. (1995) consider the learnability

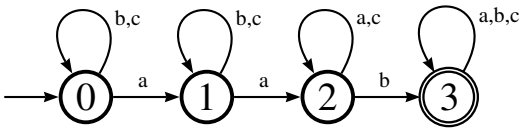

Figure 1: The canonical DFA for recognizing the shuffle ideal of $u = aab$ over $\Sigma = \{a,b,c\}$, which accepts precisely those strings that contain u as a subsequence.

of bounded-width branching programs, and show that there is an efficient algorithm to PAC learn width-2 branching programs, though not properly, and an efficient proper PAC learning algorithm for width-2 branching programs with respect to the uniform distribution. They also show that PAC learning width-3 branching programs is as hard as PAC learning DNF formulas, a problem whose status remains open.

In this paper we study the PAC learnability of another restricted class of regular languages, the shuffle ideals. The shuffle ideal generated by a string u is the collection of all strings containing u as a (not necessarily contiguous) subsequence (see Figure 1 for an illustration). Despite being a particularly simple subfamily of the regular languages, shuffle ideals play a prominent role in formal language theory. Their boolean closure forms the important family known as *piecewise-testable* languages, defined and characterized by Simon (1975). The rich structure of this language family has made it an object of intensive study, with deep connections to computability, complexity theory, and semigroups (see the papers of Lothaire (1983) and Klíma and Polák (2008) and the references therein). On a more applied front, the shuffle ideals capture some rudimentary phenomena in human language morphology (Kontorovich et al., 2003).

In Section 3 we show that shuffle ideals of known length are exactly learnable in the statistical query model under the uniform distribution, though not efficiently. Permitting approximate learning, the algorithm can be made efficient; this in turn yields efficient proper PAC learning under the uniform distribution. On the other hand, in Section 4 we show that the shuffle ideals are not properly PAC learnable under general distributions unless RP=NP. In Section 5 we show that a polynomial time improper PAC learning algorithm for the class of shuffle ideals would imply the existence of polynomial time algorithms to break the RSA cryptosystem, factor Blum integers, and test quadratic residuosity. These two negative results are analogous to those for general regular languages represented by deterministic finite automata.

2. Preliminaries

Throughout this paper, we consider a fixed finite alphabet Σ, whose size will be denoted by s. We assume $s \geq 2$. The elements of Σ^* will be referred to as *strings* with their length denoted by $|\cdot|$; the empty string is λ. The concatenation of strings u_1 and u_2 is denoted by $u_1 \cdot u_2$ or $u_1 u_2$. The string u is a *prefix* of a string v if there exists a string w such that $v = uw$. Similarly, u is a *suffix* of v if there exists a string w such that $v = wu$. We use exponential notation for repeated concatenation of a string with itself, that is, u^n is the concatenation of n copies of u.

Define the binary relation $\sqsubseteq$ on Σ^* as follows: $u \sqsubseteq v$ holds if there is a witness $\vec{i} = (i_1 < i_2 < \ldots < i_{|u|})$ such that $v_{i_j} = u_j$ for all $j \in [|u|]$. When there are several witnesses for $u \sqsubseteq v$, we may partially order them coordinate-wise, referring to the unique minimal element as the *leftmost* embedding. The unique maximal element is the *rightmost* embedding. If $u \sqsubseteq v$ then the *leftmost span* of u in v

is the shortest prefix v_1 of v such that $u \sqsubseteq v_1$ and the *rightmost span* of u in v is the shortest suffix v_2 of v such that $u \sqsubseteq v_2$.

Formally, the (principal) *shuffle ideal* generated by $u \in \Sigma^\ell$ is the regular language

$$\text{Ш}(u) = \{x \in \Sigma^* : u \sqsubseteq x\} = \Sigma^* u_1 \Sigma^* u_2 \Sigma^* \ldots \Sigma^* u_\ell \Sigma^*$$

(an example is given in Figure 1). The shuffle ideal of string u consists of all strings v over the given alphabet such that $u \sqsubseteq v$. The term *shuffle ideal* comes from algebra (Lothaire, 1983; Păun, 1994) and dates back to the paper of Eilenberg and Mac Lane (1953).

The following lemmas will be useful in the sequel. The first is immediate from the definitions; the second formalizes the obvious method of determining whether $u \sqsubseteq v$ and finding a leftmost embedding if so.

Lemma 1 *Suppose $u = u_1u_2u_3$ and $v = v_1v_2v_3$ are strings such that $u \sqsubseteq v$ and v_1 is the leftmost span of u_1 in v and v_3 is the rightmost span of u_3 in v. Then $u_2 \sqsubseteq v_2$.*

Lemma 2 *Evaluating the relation $u \sqsubseteq x$ is feasible in time $O(|x|)$.*

Proof If $u = \lambda$, then u is certainly a subsequence of x. If $u = au'$ where $a \in \Sigma$, we search for the leftmost occurrence of a in x. If there is no such occurrence, then u is certainly not a subsequence of x. Otherwise, we write $x = yax'$, where y contains no occurrence of a; then u is a subsequence of x if and only if u' is a subsequence of x', so we continue recursively with u' and x'. The total time for this algorithm is $O(|x|)$. ■

We assume a familiarity with the basics of the PAC learning model, as defined in the textbook of Kearns and Vazirani (1994). To recap, consider the instance space $\mathcal{X} = \Sigma^*$, concept class $\mathcal{C} \subseteq 2^{\mathcal{X}}$, and hypothesis class $\mathcal{H} \subseteq 2^{\mathcal{X}}$. An *algorithm* $\mathcal{L}$ is given access to a labeled sample $S = (X_i, Y_i)_{i=1}^m$, where the X_i are drawn iid from some unknown distribution P over $\mathcal{X}$ and $Y_i = f(X_i)$ for some unknown *target* $f \in \mathcal{C}$, and produces a *hypothesis* $h \in \mathcal{H}$. We say that $\mathcal{L}$ efficiently PAC learns $\mathcal{C}$ if for any $\varepsilon, \delta > 0$ there is an $m_0 \in \mathbb{N}$ such that for all $f \in \mathcal{C}$ and all distributions P, the hypothesis h_m generated by $\mathcal{L}$ based on a sample of size $m \geq m_0$ satisfies

$$P^m[P(\{x \in \mathcal{X} : h_m(x) \neq f(x)\}) > \varepsilon] < \delta;$$

moreover, we require that both m_0 and $\mathcal{L}$'s runtime be at most polynomial in ε^{-1}, δ^{-1} and the sizes of f and X_i. The learning is said to be *proper* if $\mathcal{H} = \mathcal{C}$ and *improper* otherwise. If the learning algorithm achieves $\varepsilon = 0$, the learning is said to be *exact* (Bshouty, 1997; Bshouty et al., 2005).

Most learning problems can be cleanly decomposed into a computational and an information theoretic component. The information theoretic aspects of learning automata are well understood. As mentioned above, the VC-dimension of a collection of DFAs grows polynomially with maximal number of states, and so any small DFA consistent with the training sample will, with high probability, have small generalization error. For shuffle ideals, an even simpler bound can be derived. If n is an upper bound on the length of the string $u \in \Sigma^*$ generating the target shuffle ideal, then our concept class contains exactly

$$\sum_{\ell=0}^{n} |\Sigma|^\ell = O(|\Sigma|^n)$$

members. Thus, with probability at least $1-\delta$, any shuffle ideal consistent with a sample of size m will achieve a generalization error of

$$O\left(\frac{n\log|\Sigma|-\log\delta}{m}\right).$$

Hence, the problem of properly PAC learning shuffle ideals has been reduced to finding one that is consistent with a given sample. This is shown to be computationally hard under adversarial distributions (Theorem 7), but feasible under the uniform one (Theorem 6). Actually, our positive result is somewhat stronger: since we show learnability in the statistical query (SQ) model of Kearns (1998), this implies a noise tolerant PAC result. In addition, in Section 5 we show that the existence of a polynomial time improper PAC learning algorithm for shuffle ideals would imply the existence of polynomial time algorithms for certain cryptographic problems.

3. SQ Learning Under the Uniform Distribution

The main result of this section is that shuffle ideals are efficiently PAC learnable under the uniform distribution. To be more precise, we are dealing with the instance space $X=\Sigma^n$ endowed with the uniform distribution, which assigns a weight of $|\Sigma|^{-n}$ to each element of X. Our learning algorithm is most naturally expressed in the language of *statistical queries* (Kearns, 1998; Kearns and Vazirani, 1994). In the original definition, a statistical query χ is a binary predicate of a random instance-label pair, and the oracle returns the value $\mathbf{E}\chi$, additively perturbed by some amount not exceeding a specified tolerance parameter. We will consider a somewhat richer class of queries.

3.1 Constructing and Analyzing the Queries

For $u\in\Sigma^{\le n}$ and $a\in\Sigma$, we define the query $\chi_{u,a}(\cdot,\cdot)$ by

$$\chi_{u,a}(x,y)=\begin{cases}0, & u\not\sqsubseteq x'\\ y(\mathbb{1}_{\{\sigma=a\}}-\mathbb{1}_{\{\sigma\neq a\}}/(s-1)), & u\sqsubseteq x'\end{cases}$$

where x' is the prefix of x of length $(n-1)$, σ is the symbol in x following the leftmost embedding of u and $\mathbb{1}_{\{\pi\}}$ represents the 0-1 truth value of the predicate π (recall that $s=|\Sigma|$). Our definition of the query $\chi_{u,a}$ is legitimate because (i) it can be efficiently evaluated (Lemma 2) and (ii) it can be expressed as a linear combination of $O(1)$ standard binary queries (also efficiently computable). In words, the function $\chi_{u,a}$ computes the mapping $(x,y)\mapsto\mathbb{R}$ as follows. If u is not a subsequence of x', $\chi_{u,a}(x,y)=0$. Otherwise, $\chi_{u,a}$ checks whether the symbol σ in x following the leftmost embedding of u is equal to a, and, if x is a positive example ($y=+1$), returns 1 if $\sigma=a$, or $-1/(s-1)$ if $\sigma\neq a$. If x is a negative example ($y=-1$) then the signs of the values returned are inverted.

Suppose for now that the length $L=|\bar{u}|$ of the target shuffle ideal $\bar{u}$ is known. Our learning algorithm uses statistical queries to recover $\bar{u}\in\Sigma^L$ one symbol at a time. It starts with the empty string $u=\lambda$. Having recovered $u=\bar{u}_1,\ldots,\bar{u}_\ell$, $\ell<L$, we infer $\bar{u}_{\ell+1}$ as follows. For each $a\in\Sigma$, the SQ oracle is called with the query $\chi_{u,a}$ and a tolerance $0<\tau<1$ to be specified later. Our key technical observation is that the value of $\mathbf{E}\chi_{u,a}$ effectively selects the next symbol of $\bar{u}$:

Lemma 3

$$\mathbf{E}\chi_{u,a}=\begin{cases}+\frac{2}{s}P(L,n,s), & a=\bar{u}_{\ell+1}\\ -\frac{2}{s(s-1)}P(L,n,s), & a\neq\bar{u}_{\ell+1}\end{cases}$$

where

$$P(L,n,s) = \binom{n-1}{L-1}\left(\frac{1}{s}\right)^{L-1}\left(1-\frac{1}{s}\right)^{n-L}.$$

Proof Fix an unknown string $\bar{u}$ of length $L \geq 1$; by assumption, we have recovered in $u = u_1 \ldots u_\ell = \bar{u}_1 \ldots \bar{u}_\ell$ the first ℓ symbols of $\bar{u}$. Let $u' = \bar{u}0^\infty$ be the extension of $\bar{u}$ obtained by padding it on the right with infinitely many 0 symbols (we assume $0 \in \Sigma$).

Let X be a random variable representing the uniformly chosen sample string x. Let T be the largest value for which $u'_1 \ldots u'_T$ is a subsequence of X. Let $\xi = \mathbb{1}_{\{T \geq L\}}$ be the indicator for the event that X is a positive instance, that is, that $\bar{u}_1 \ldots \bar{u}_L = u'_1 \ldots u'_L$ is a subsequence of X.

Observe that T has a binomial distribution:

$$T \sim \text{Binom}(n, 1/s);$$

indeed, as we sweep across X, each position X_i has a $1/s$ chance of being the next unused symbol of u'. An immediate consequence of this fact is that $\Pr[\xi = 1]$ is exactly $\sum_{k=L}^{n}\binom{n}{k}(1/s)^k(1-1/s)^{n-k}$.

Now fix $\ell < L$ and let I_ℓ be defined as follows. If $\ell = 0$ then $I_\ell = 0$, and if $u_1 \ldots u_\ell$ is not a subsequence of $X_1 \ldots X_{n-1}$ then $I_\ell = n-1$. Otherwise, I_ℓ is the position of u_ℓ in the leftmost embedding of $u_1 \ldots u_\ell$ in $X_1 \ldots X_{n-1}$. Then $I_\ell + 1$ is the position of σ as defined in (3.1), or n if $u_1 \ldots u_\ell \not\sqsubseteq X_1 \ldots X_{n-1}$.

We define two additional random variables, T_A and T_B. T_A is the length of the longest prefix of u' that is a subsequence of X with $X_{I_\ell+1}$ excluded:

$$T_A = \max\left\{t : u'_1 \ldots u'_t \sqsubseteq X_1 \ldots X_{I_\ell} X_{I_\ell+2} \ldots X_n\right\}.$$

Intuitively, T_B is the length of the longest prefix of u' with $u'_{\ell+1}$ excluded that is a subsequence of X with $X_{I_\ell+1}$ excluded. Formally, let $v_1 v_2 \ldots$ be the sequence $u'_1 u'_2 \ldots$ with the element $u'_{\ell+1}$ excluded, that is, $v_i = u'_i$ if $i \leq \ell$ and $v_i = u'_{i+1}$ if $i \geq \ell+1$.

$$T_B = \max\left\{t : v_1 \ldots v_t \sqsubseteq X_1 \ldots X_{I_\ell} X_{I_\ell+2} \ldots X_n\right\}.$$

Like T, T_A and T_B are binomially distributed, but now

$$T_A, T_B \sim \text{Binom}(n-1, 1/s).$$

The reason is that we always omit one position in X (the one following u_ℓ if u_ℓ appears before X_n or X_n if it does not), and for each other position, there is still an independent $1/s$ chance that it is the next symbol in u' (or u' with $u'_{\ell+1}$ excluded.)

An important fact is that $X_{I_\ell+1}$ is independent of the values of T_A and T_B, though of course T_A and T_B are not independent of each other. This is not immediately obvious: whether $X_{I_\ell+1}$ equals $u'_{\ell+1}$ or not affects the interpretation of later symbols in X. However, the probability that each symbol $X_{I_\ell+2} \ldots$ is the next unused symbol in u' (or v) is still an independent $1/s$ whether $X_{I_\ell+1}$ consumes a symbol of u' (or v) or not. The joint distribution of T_A and T_B is not affected.

We now compute $\mathbf{E}\chi_{u,a}$ by averaging over the choices in the joint distribution of T_A and T_B. If $T_A \geq L$, then $\bar{u}$ is a subsequence of $X_1 \ldots X_{I_\ell} X_{I_\ell+2} \ldots X_n$, and X is a positive example ($y = +1$) no matter how $X_{I_\ell+1}$ is chosen. In this case, each symbol in Σ contributes 1 to the conditional expected value with probability $1/s$ and $-\frac{1}{s-1}$ with probability $\frac{s-1}{s}$; the net contribution is 0.

If X is a positive example, then $\bar{u}$ is a subsequence of X and a leftmost embedding of $\bar{u}$ in X embeds $u_1 \dots u_\ell$ in $X_1 \dots X_{I_\ell}$ and embeds $u_{\ell+1} \dots u_L$ in $X_{I_\ell+1} \dots X_n$. Thus, no matter what symbol is chosen for $X_{I_\ell+1}$, $u_{\ell+2} \dots u_L$ is a subsequence of $X_{I_\ell+2} \dots X_n$, and T_B must be at least $L-1$. Thus, if $T_A \geq L$ then $T_B \geq L-1$. Moreover, if $T_B < L-1$, X must be a negative example ($y=-1$) no matter how $X_{I_\ell+1}$ is chosen. In this case, the probability-$(1/s)$ contribution of -1 is exactly offset by the probability-$\left(\frac{s-1}{s}\right)$ contribution of $\frac{1}{s-1}$, and the conditional expected value is 0.

Thus the only case in which there may be a non-zero contribution to the expected value is when $T_A < L$ and $T_B \geq L-1$, that is, when the choice of $X_{I_\ell+1}$ may affect the label of X. The example X is positive if and only if $X_{I_\ell+1} = \bar{u}_{\ell+1}$, which occurs if $\sigma = \bar{u}_{\ell+1}$. Thus the conditional expectation for $a = \bar{u}_{\ell+1}$ is

$$1 \cdot \Pr[\sigma = \bar{u}_{\ell+1}] + \frac{1}{s-1} \cdot \Pr[\sigma \neq \bar{u}_{\ell+1}] = \frac{1}{s} + \frac{1}{s-1} \cdot \frac{s-1}{s} = 2/s.$$

For $a \neq \bar{u}_{\ell+1}$, the conditional expectation is is $-\frac{2}{s(s-1)}$. This can be computed directly by considering cases, or by observing that the change to $\sum_{a \in \Sigma} \chi_{u,a}(x) = 0$ always, and that all $a \neq \bar{u}_{\ell+1}$ induce same expectation by symmetry.

Finally we need to determine $\Pr[T_A < L \wedge T_B \geq L-1]$. We may write

$$\Pr[T_B \geq L-1 \wedge T_A < L] = \Pr[T_B \geq L-1] - \Pr[T_B \geq L-1 \wedge T_A \geq L]$$

because $T_A \geq L$ implies $T_B \geq L-1$,

$$\Pr[T_B \geq L-1 \wedge T_A \geq L] = \Pr[T_A \geq L],$$

and thus

$$\Pr[T_B \geq L-1 \wedge T_A < L] = \Pr[T_B \geq L-1] - \Pr[T_A \geq L].$$

Because T_A and T_B are binomially distributed, $\Pr[T_B \geq L-1 \wedge T_A < L]$ is

$$\sum_{i=L-1}^{n-1} \binom{n-1}{i} \left(\tfrac{1}{s}\right)^i \left(1-\tfrac{1}{s}\right)^{n-1-i} - \sum_{i=L}^{n-1} \binom{n-1}{i} \left(\tfrac{1}{s}\right)^i \left(1-\tfrac{1}{s}\right)^{n-1-i}$$

which is

$$\binom{n-1}{L-1} \left(\tfrac{1}{s}\right)^{L-1} \left(1-\tfrac{1}{s}\right)^{n-L} = P(L,n,s).$$

This concludes the proof of Lemma 3. ■

3.2 Specifying the Query Tolerance τ

The analysis in Lemma 3 implies that to identify the next symbol of $\bar{u} \in \Sigma^L$ it suffices to distinguish the two possible expected values of $\mathbf{E}\chi_{u,a}$, which differ by $(2/(s-1))P(L,n,s)$. If the query tolerance is set to one third of this value, that is,

$$\tau = \frac{2}{3(s-1)} P(L,n,s)$$

then s statistical queries for each prefix of $\bar{u}$ suffice to learn $\bar{u}$ exactly.

Theorem 4 *When the length L of the target string $\bar{u}$ is known, $\bar{u}$ is exactly identifiable with $O(Ls)$ statistical queries at tolerance $\tau = \frac{2}{3(s-1)}P(L,n,s)$.*

In the above SQ algorithm there is no need for a precision parameter ε because the learning is *exact*, that is, $\varepsilon = 0$. Nor is there a need for a confidence parameter δ because each statistical query is guaranteed to return an answer within the specified tolerance, in contrast to the PAC setting where the parameter δ protects the learner against an "unlucky" sample.

However, if the relationship between n and L is such that $P(L,n,s)$ is very small, then the tolerance τ will be very small, and this first SQ algorithm cannot be considered efficient. If we allow an approximately correct hypothesis ($\varepsilon > 0$), we can modify the above algorithm to use a polynomially bounded tolerance.

Theorem 5 *When the length L of the target string $\bar{u}$ is known, $\bar{u}$ is approximately identifiable to within $\varepsilon > 0$ with $O(Ls)$ statistical queries at tolerance $\tau = 2\varepsilon/(9(s-1)n)$.*

Proof We modify the SQ algorithm to make an initial statistical query with tolerance $\varepsilon/3$ to estimate $\Pr[\xi = 1]$, the probability that x is a positive example. If the answer is $\leq 2\varepsilon/3$, then $\Pr[\xi = 1] \leq \varepsilon$ and the algorithm outputs a hypothesis that classifies all examples as negative. If the answer is $\geq 1 - 2\varepsilon/3$, then $\Pr[\xi = 1] \geq 1 - \varepsilon$ and the algorithm outputs a hypothesis that classifies all examples as positive.

Otherwise, $\Pr[\xi = 1]$ and $\Pr[\xi = 0]$ are both at least $\varepsilon/3$, and the first SQ algorithm is used. We now show that $P(L,n,s) \geq \varepsilon/(3n)$, establishing the bound on the tolerance. Let $Q(L,n,s) = \binom{n}{L}\left(\frac{1}{s}\right)^L\left(1-\frac{1}{s}\right)^{n-L}$ and note that $Q(L,n,s) = (n/Ls)P(L,n,s)$. If $L \leq n/s$ then $Q(L,n,s)$ is at least as large as every term in the sum

$$\Pr[\xi = 0] = \sum_{k=0}^{L-1} \binom{n}{k} \left(\frac{1}{s}\right)^k \left(1-\frac{1}{s}\right)^{n-k}$$

and therefore $Q(L,n,s) \geq \varepsilon/(3L)$ and $P(L,n,s) \geq \varepsilon/(3n)$. If $L > n/s$ then $Q(L,n,s)$ is at least as large as every term in the sum

$$\Pr[\xi = 1] = \sum_{k=L}^{n} \binom{n}{k} \left(\frac{1}{s}\right)^k \left(1-\frac{1}{s}\right)^{n-k}$$

and therefore $P(L,n,s) \geq Q(L,n,s) \geq \varepsilon/(3n)$. ■

3.3 PAC Learning

The main result of this section is now obtained by a standard transformation of an SQ algorithm to a PAC algorithm.

Theorem 6 *The concept class $\mathcal{C} = \left\{\text{Ш}(u) : u \in \Sigma^{\leq n}\right\}$ is efficiently properly PAC learnable under the uniform distribution.*

Proof We assume that the algorithm receives as inputs n, L, ε and δ. Because there are only $n+1$ choices of L, a standard method may be used to iterate through them. We simulate the modified SQ

algorithm by drawing a sample of labeled examples and using them to estimate the answers to the $O(Ls)$ calls to the SQ oracle with queries at tolerance $\tau = 2\varepsilon/(9(s-1)n)$, as described by Kearns (1998). According to the result of Kearns (1998, Theorem 1),

$$O\left(\frac{1}{\tau^2}\log\frac{|C|}{\delta}\right) = O\left(\frac{s^2n^2}{\varepsilon^2}(n\log s - \log\delta)\right)$$

examples suffice to determine correct answers to all the queries at the desired tolerance, with probability at least $1-\delta$. ■

Our learning algorithm and analysis are rather strongly tied to the uniform distribution. If this assumption is omitted, it might now happen that $\Pr[T_B \geq L-1 \wedge T_A < L]$ is small even though positive and negative examples are mostly balanced, or there might be intractable correlations between σ and the values of T_A and T_B. It seems that genuinely new ideas will be required to handle nonuniform distributions.

4. Proper PAC Learning Under General Distributions Is Hard Unless NP=RP

This hardness result follows a standard paradigm (see Kearns and Vazirani, 1994). We show that the problem of deciding whether a given labeled sample admits a consistent shuffle ideal is NP-complete. A standard argument then shows that any proper PAC learner for shuffle ideals can be efficiently manipulated into solving the decision problem, yielding an algorithm in RP. Thus, assuming RP $\neq$ NP, there is no polynomial time algorithm that properly learns shuffle ideals.

Theorem 7 *For any alphabet of size at least 2, given two disjoint sets of strings $S, T \subset \Sigma^*$, the problem of determining whether there exists a string u such that $u \sqsubseteq x$ for each $x \in S$ and $u \not\sqsubseteq x$ for each $x \in T$ is NP-complete.*

We first prove a lemma that facilitates the representation of n independent binary choices. Let $\Sigma = \{0,1\}$, let n be a positive integer and define A_n to be the set of 2^n binary strings described by the regular expression

$$((00000 + 00100)11)^n.$$

Define strings

$$\begin{aligned} v_0 &= 000100, \\ v_1 &= 001000, \\ d &= 11, \end{aligned}$$

and let S_n consist of the two strings

$$\begin{aligned} s_0 &= (v_0 d)^n, \\ s_1 &= (v_1 d)^n. \end{aligned}$$

Define the strings

$$\begin{aligned} y_0 &= 00010, \\ y_1 &= 01000, \\ z &= 0000, \\ d_0 &= 1 \end{aligned}$$

and for each integer i such that $1 \leq i \leq n$, define the strings

$$\begin{aligned} t_{i,0} &= (v_0 d)^{i-1} y_0 d (v_0 d)^{n-i}, \\ t_{i,1} &= (v_0 d)^{i-1} y_1 d (v_0 d)^{n-i}, \\ t_{i,2} &= (v_0 d)^{i-1} z d (v_0 d)^{n-i}, \\ t_{i,3} &= (v_0 d)^{i-1} v_0 d_0 (v_0 d)^{n-i}. \end{aligned}$$

The strings $t_{i,0}$, $t_{i,1}$ and $t_{i,2}$ are obtained from s_0 by replacing occurrence i of v_0 by y_0, y_1, and z, respectively. The string $t_{i,3}$ is obtained from s_0 by replacing occurrence i of d by d_0. Let T_n consist of all the strings $t_{i,j}$ for $1 \leq i \leq n$ and $0 \leq j \leq 3$.

The following lemma shows that the set of strings consistent with S_n and T_n is precisely the 2^n strings in A_n.

Lemma 8 *Let C_n be the set of strings u such that u is a subsequence of both strings in S_n and not a subsequence of any string in T_n. Then $C_n = A_n$.*

Proof We first observe that for any positive integer m and any string $u \in A_m$, the leftmost span of u in $(v_0 d)^m$ is $(v_0 d)^m$ itself, and the leftmost span of u in $(v_1 d)^m$ is $(v_1 d)^m$ itself. For $m = 1$, we have $u = 0000011$ or $u = 0010011$, while $v_0 d = 00010011$ and $v_1 d = 00100011$, and the result holds by inspection. Then a straightforward induction establishes the result for $m > 1$. Similarly, for any string $u \in A_m$, the rightmost span of du in $d(v_0 d)^m$ is $d(v_0 d)^m$ itself, and the rightmost span of du in $d(v_1 d)^m$ is $d(v_1 d)^m$ itself. In the base case we have $du = 110000011$ or $du = 110010011$, while $dv_0 d = 1100010011$ and $dv_1 d = 1100100011$, and the result holds by inspection. A straightforward induction establishes the result for $m > 1$.

Suppose $u \in A_n$. Then

$$u = u_1 d u_2 d \cdots u_n d,$$

where each u_i is either 00000 or 00100. Clearly $u \sqsubseteq s_0$ and $u \sqsubseteq s_1$, because 00000 and 00100 are subsequences of v_0 and v_1.

Consider a string $t_{i,0} \in T_n$. Suppose that $u \sqsubseteq t_{i,0}$. Divide u into three parts, $u = u' u_i u''$, where u' is $u_1 d \cdots u_{i-1} d$ and $u'' = d u_{i+1} \cdots u_n d$. The leftmost span of u' in $t_{i,0}$ is $(v_0 d)^{i-1}$, and the rightmost span of u'' in $t_{i,0}$ is $d(v_0 d)^{n-i}$, which implies that $u_i \sqsubseteq y_0$ by Lemma 1. But u_i is either 00000 or 00100 and y_0 is 00010, which is a contradiction. So u is not a subsequence of $t_{i,0}$. Similar arguments show that u is not a subsequence of $t_{i,1}$ or $t_{i,2}$.

Now suppose $u \sqsubseteq t_{i,3}$. We divide u into parts, $u = u' u_i d u_{i+1} u''$, where $u' = u_1 d \cdots u_{i-1} d$ and $u'' = d u_{i+2} \cdots u_n d$. The leftmost span of u' in $t_{i,3}$ is $(v_0 d)^{i-1}$ and the rightmost span of u'' in $t_{i,3}$ is $d(v_0 d)^{n-i-1}$. By Lemma 1, we must have

$$u_i d u_{i+1} \sqsubseteq v_0 d_0 v_0.$$

That is, at least one of the strings

$$000001100000, 001001100000, 000001100100, 001001100100$$

must be a subsequence of 0001001000100, which is false, showing that u is not a subsequence of $t_{i,3}$. Thus u is not a subsequence of any string in T_n, and $u \in C_n$. Thus $A_n \subseteq C_n$.

For the reverse direction, suppose $u \in C_n$. We consider an embedding of u in s_0 and divide u into segments

$$u = u_1 d_1 u_2 d_2 \cdots u_n d_n,$$

where for each i, $u_i \sqsubseteq v_0$ and $d_i \sqsubseteq d$. If for any i we have $d_i \sqsubseteq 1$, then $u \sqsubseteq t_{i,3}$, a contradiction. Thus $d_i = 11 = d$ for every i. Similarly, if u_i is a subsequence of y_0, y_1 or z, then u is a subsequence of $t_{i,0}$, $t_{i,1}$, or $t_{i,2}$, respectively, so we know that each u_i is a subsequence of the string 000100, but not a subsequence of the strings 00010, 01000, or 0000. It is not difficult to check that the only possibilities for u_i are

$$00000, 00100, 000100.$$

To eliminate the third possibility we use the fact that u is a subsequence of s_1. Consider any string

$$w = w_1 d w_2 d \cdots w_n d,$$

where $w_i = 000100$ and each w_j for $j \neq i$ is either 00000 or 00100. We may divide w into parts $w = w'000100w''$ where $w' = w_1 d \cdots w_{i-1} d$ and $w'' = d w_{i+1} d \cdots w_n d$. If $w \sqsubseteq s_1$, then the leftmost span of w' in s_1 is $(v_1 d)^{i-1}$, and the rightmost span of w'' in s_1 is $d(v_1 d)^{n-i}$, which by Lemma 1 means that 000100 must be a subsequence of $v_1 = 001000$, a contradiction. Thus no such w is a subsequence of s_1, and we must have u_i equal to 00000 or 00100 for all i, that is, u must be in A_n. Thus $C_n \subseteq A_n$. ■

We now prove Theorem 7.

Proof To see that this decision problem is in NP, note that if S is empty, then any string of length longer than the longest string in T satisfies the necessary requirements, so that the answer in this case is necessarily "yes." If S is nonempty, then no string longer than the shortest string in S can be a subsequence of every string in S, so we need only guess a string w whose length is bounded by that of the shortest string in S and check whether w is a subsequence of every string in S and of no string in T, which takes time proportional to the sum of the lengths of all the input strings (Lemma 2).

To see that this problem is complete in NP, we reduce satisfiability of CNF formulas to this question. Given a CNF formula ϕ over the n variables x_i for $1 \leq i \leq n$, we construct two sets of binary strings S and T such that ϕ is satisfiable if and only if there exists a shuffle string u that is a subsequence of every string in S and of no string in T. The set S is just the two strings s_0 and s_1 in the set S_n. The set T is the strings in the set T_n together with additional strings determined by the clauses of ϕ. By Lemma 8, the strings consistent with S_n and T_n are the 2^n strings in A_n.

We use each $u = u_1 d u_2 d \cdots u_n d$ in A_n to represent an assignment to the n variables x_i by choosing $x_i = 0$ if u_i is 00000 and $x_i = 1$ if $u_i = 00100$. We construct additional elements of T based on the clauses of the formula ϕ to exclude any strings representing assignments that do not satisfy ϕ. For example, if clause j of ϕ is

$$(x_3 \vee \overline{x}_6 \vee \overline{x}_{17}),$$

we add a string t_j to T obtained from s_0 by replacing occurrence 3 of v_0 by 00000, replacing occurrence 6 of v_0 by 00100, and occurrence 17 of v_0 by 00100, where we have chosen 00000 or 00100 to falsify the corresponding literal. The strings in A_n that are subsequences of t_j are exactly those that correspond to assignments that falsify clause j of ϕ, and adding t_j to T eliminates these strings from those consistent with S and T. By adding one string t_j to T for each clause j of ϕ, we ensure that the only strings u that are subsequences of both elements of S and not subsequences of any element of T are exactly those elements of A_n that correspond to assignments that do not falsify any clause of ϕ. Thus, there exists at least one string u that is a subsequence of both strings in S and not a subsequence of any string in T if and only if ϕ is satisfiable.

Note that S contains two strings of length $O(n)$, T_n contains $4n$ strings of length $O(n)$, and T additionally contains one string of length $O(n)$ for each clause of ϕ, so the sizes of S and T are polynomial in the size of ϕ. This completes the proof of Theorem 7. ■

5. Cryptographic Limitations on PAC Learning Shuffle Ideals

In this section we show that the problem of PAC learning any class of constant-depth, polynomial-size threshold formulas is efficiently reducible to the problem of PAC learning shuffle ideals. Because for some constant depth, the class of polynomial-size threshold formulas of that depth are capable of computing iterated product, the results of Kearns and Valiant (1994) imply that a polynomial time PAC algorithm to learn them would imply polynomial time algorithms for certain fundamental problems in cryptography, namely, inverting RSA encryption, factoring Blum integers, and testing quadratic residuosity. Thus, the class of shuffle ideals faces the same cryptographic limitations on PAC learnability as demonstrated by Kearns and Valiant for the class of general regular languages represented by deterministic finite automata.

A *threshold function* is a Boolean function with m inputs and a threshold t. Its output is 1 if at least t of its inputs are 1 and 0 otherwise. Thus, an OR of m inputs is equivalent to a threshold function with threshold 1, and an AND of m inputs is equivalent to a threshold function with threshold m. There are $m+2$ different threshold functions of m inputs, corresponding to $t = 0, 1, \ldots, m+1$. The threshold $t = 0$ computes the constant function 1, while the threshold $t = m+1$ computes the constant function 0.

Given an integer $m > 1$, we define the class $T(n,m,d)$ of threshold formulas over the variables $V_n = \{x_1, x_2, \ldots, x_n\}$ of fan-in exactly m and depth d by induction on d as follows. The formulas of depth $d = 0$ are the two constants 0 and 1 and the $2n$ literals x_i and $\overline{x}_i$. For $d > 0$, the formulas of depth d consist of a threshold function with m inputs applied to a sequence of m formulas of depth $d-1$. Note that a threshold function of m inputs can be used to compute a threshold function of fewer inputs by insuring that the excess inputs are the constant function 0.

We can picture the elements of $T(n,m,d)$ as ordered full m-ary trees of depth d whose internal nodes are labeled by threshold functions, and whose leaves are labeled by constants or literals. Thus, the total number of occurrences of constants or literals in a threshold formula of fan-in m and depth d is $O(m^d)$. If d is a fixed constant and m is bounded by a polynomial in n, the total size of such a formula is bounded by a polynomial in n. The same is true if m is a fixed constant and d is bounded by $O(\log n)$; in this case, the formulas compute functions in the class NC1 of constant fan-in, logarithmic depth Boolean circuits.

We now describe a reduction parameterized by d that maps each threshold formula f in $T(n,m,d)$ to a shuffle string $r_d(f)$, and each assignment a to the variables V_n to an assignment string $s_d(a)$, such that the assignment a satisfies f if and only if the shuffle string $r_d(f)$ is a subsequence of the assignment string $s_d(a)$. The string alphabet consists of the symbols 0 and 1 and a set of $d+1$ delimiters: $\#_0, \#_1, \ldots, \#_d$.

The base case is $d = 0$, where f is a single constant 0 or 1 or a single literal x_i or $\bar{x}_i$. In this case, the shuffle string is

$$r_0(f) = y_1 \#_0 y_2 \#_0 \ldots y_n \#_0,$$

where y_j is defined as follows. If $f = 0$ then $y_j = 01$ for all j, and if $f = 1$ then $y_j = \lambda$ for all j. If $f = x_i$ then $y_j = \lambda$ for all $j \neq i$ and $y_i = 1$, while if $f = \bar{x}_i$ then $y_j = \lambda$ for all $j \neq i$ and $y_i = 0$.

If the assignment a is given by a binary string $a_1 a_2 \ldots a_n$, indicating that x_i is assigned the value a_i, then the string representing the assignment is just

$$s_0(a) = a_1 \#_0 a_2 \#_0 \ldots a_n \#_0.$$

It is clear that $r_0(f)$ is a subsequence of $s_0(a)$ if and only if the n occurrences of $\#_0$ in each string are matched, and y_j is a subsequence of a_j for all $j = 1, 2, \ldots, n$. For $f = 0$ we have $y_j = 01$ for all j, so this holds for no a. For $f = 1$ we have $y_j = \lambda$ for all j, and this holds for every a. If f is a literal, then this holds if and only if $y_i = a_i$, that is, if and only if a satisfies f. Thus, when f is a constant or a literal, $r_0(f)$ is a subsequence of $s_0(a)$ if and only if a satisfies f.

In addition to defining the shuffle string and the assignment strings at each level, we also define a slack string. For level 0, the slack string z_0 is defined as follows.

$$z_0 = (01\#_0)^n,$$

That is, z_0 consists of n repetitions of the string $01\#_0$. For level d, the slack string is designed to ensure that $r_d(f)$ is a subsequence of z_d for any $f \in T(n,m,d)$; this clearly holds at level $d = 0$.

For the inductive case $d > 0$, we assume that the construction has been defined for $d-1$ using symbols 0, 1, and delimiters $\#_0, \ldots, \#_{d-1}$. Thus the level d delimiter, $\#_d$, has not yet been used. Suppose f is a depth d threshold formula from $T(n,m,d)$, that is,

$$f = \theta(f_1, f_2, \ldots, f_m),$$

where each f_i is a depth $d-1$ threshold formula and θ is a threshold function with threshold t. We define the shuffle string

$$r_d(f) = u_1 u_1 u_2 u_2 \cdots u_m u_m (\#_d)^{2t},$$

where for each $i = 1, 2, \ldots, m$,

$$u_i = r_{d-1}(f_i) \#_d.$$

That is, $r_d(f)$ consists of two copies of the level $d-1$ code for f_i, with each copy followed by the delimiter $\#_d$, for $i = 1, 2, \ldots, m$, followed by t pairs of the delimiter $\#_d$. Note that $r_d(f)$ may contain up to $4m+2$ copies of $\#_d$.

Given an assignment a to the variables V_n, we define a level d assignment string

$$s_d(a) = v^{2m},$$

where

$$v = s_{d-1}(a) \#_d z_{d-1} \#_d.$$

That is, $s_d(a)$ is $2m$ copies of the string v consisting of the level $d-1$ code for a, followed by $\#_d$, followed by the level $d-1$ slack string, followed by $\#_d$. Note that $s_d(a)$ contains exactly $4m$ copies of $\#_d$.

Finally, the level d slack string is defined as follows.

$$z_d = (z_{d-1}\#_d)^{4m+2}.$$

A straightforward induction shows that for any threshold formula f in $T(n,m,d)$, $r_d(f)$ is a subsequence of z_d, and for any assignment a to the variables, $s_d(a)$ is also a subsequence of z_d.

Lemma 9 *For all threshold formulas f in $T(n,m,d)$ and assignments a to the variables in V_n, a satisfies f if and only if $r_d(f)$ is a subsequence of $s_d(a)$.*

Proof This is proved by induction on d. For $d = 0$, the basis construction showed that for all constants or literals f and assignments a, a satisfies f if and only if $r_0(f)$ is a subsequence of $s_0(a)$.

Inductively assume that the construction works for $d-1$. Suppose f is a depth d threshold formula, that is,

$$f = \theta(f_1, f_2, \ldots, f_m),$$

where each f_i is a depth $d-1$ threshold formula and θ is a threshold function with threshold t. For any index i and any assignment a let

$$u_i = r_{d-1}(f_i)\#_d$$

and

$$v = s_{d-1}(a)\#_d z_{d-1}\#_d.$$

Because $r_{d-1}(f_i)$ is a subsequence of the slack string z_{d-1}, u_iu_i is a subsequence of vv. Also, u_iu_i is a subsequence of v if and only if $r_{d-1}(f_i)$ is a subsequence of $s_{d-1}(a)$, which holds if and only if a satisfies f_i, by the inductive assumption. If u_iu_i is not a subsequence of v, then a leftmost embedding of u_iu_i in vv must match the first $\#_d$ in u_iu_i to the second $\#_d$ in vv and the second $\#_d$ in u_iu_i to the fourth $\#_d$ in vv, thereby "consuming" all of vv for the embedding.

Suppose a satisfies f. Because θ is a threshold function with threshold t, there must be a set T of at least t indices i such that a satisfies f_i. By the inductive assumption, this means that $r_{d-1}(f_i)$ is a subsequence of $s_{d-1}(a)$ for each $i \in T$. For each $i \in T$, u_iu_i is a subsequence of v. For $i \notin T$, u_iu_i is a subsequence of vv but not of v. Thus we can find a leftmost embedding of $r_d(f)$ in $s_d(a)$ by consuming one copy of v from $s_d(a)$ for each $i \in T$ and two copies for each $i \notin T$, using at most $2m-t$ copies, and leaving at least t copies, which allows us to embed the trailing sequence of $2t$ delimiters $\#_d$ in the remaining copies of v. Thus $r_d(f)$ is a subsequence of $s_d(a)$.

Conversely, suppose that $r_d(f)$ is a subsequence of $s_d(a)$, and consider a leftmost embedding. Considering the segments u_iu_i of $r_d(f)$ from left to right, we see that the leftmost embedding consumes one copy of v if a satisfies f_i and two copies if a does not satisfy f_i. Thus, if T is the set of indices i such that a satisfies f_i, then after embedding all m such segments, $2m-|T|$ copies of v are consumed from $s_d(a)$, leaving $|T|$ copies. Because the trailing $2t$ occurrences of $\#_d$ in $r_d(f)$ are matched in the remaining portion of $s_d(a)$, we must have $2|T| \geq 2t$, and therefore a satisfies f_i for at least t indices i, that is, a satisfies f. ■

How long are the strings $r_d(f)$ and $s_d(a)$? Each is a subsequence of z_d, and for $m \geq 2$, the length of z_d is bounded by $(10m)^d(3n)$. This is polynomial in n if either d is a fixed constant and m

is polynomial in n, or if m is a fixed constant and $d = O(\log n)$. In either case, the mapping from a to $s_d(a)$ is computable in polynomial time, and we have the following results.

The first result assumes a polynomial time algorithm to learn shuffle ideals over some fixed alphabet.

Theorem 10 *Suppose for some positive integer d, there exists a polynomial time algorithm to PAC learn shuffle ideals over an alphabet of size $d+2$. Then for any polynomial $p(n)$, there exists a polynomial time algorithm to PAC learn the threshold formulas in $T(n, p(n), d)$.*

The second result assumes a polynomial time algorithm to learn shuffle ideals over an arbitrary finite alphabet, where the dependence on the alphabet size must be at most exponential.

Theorem 11 *Suppose there exists an algorithm to PAC learn shuffle ideals over arbitrary finite alphabets that runs in time polynomial in n and C^s, where n is a bound on the length of examples, s is the alphabet size and C is a fixed constant. Then for any constant K, there exists a polynomial time algorithm to PAC learn the threshold formulas in $T(n, 2, K\log n)$.*

5.1 Example of the Construction of $r_d(f)$ and $s_d(a)$

We illustrate the construction for the formula

$$f = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \wedge x_3)$$

from $T(3,2,2)$ and the assignment $a = 001$. To avoid subscripted delimiters, let #, \$, and % stand for $\#_0$, $\#_1$ and $\#_2$ respectively. For the base case we have the following.

$$\begin{aligned} r_0(x_1) &= 1\#\#\#, \\ r_0(\bar{x}_1) &= 0\#\#\#, \\ r_0(\bar{x}_2) &= \#0\#\#, \\ r_0(x_3) &= \#\#1\#, \\ z_0 &= 01\#01\#01\#. \end{aligned}$$

The two subformulas of f have thresholds of 1 and 2 respectively.

$$\begin{aligned} r_1(x_1 \vee \bar{x}_2) &= 1\#\#\#\$1\#\#\#\$\#0\#\#\$\#0\#\#\$\$\$, \\ r_1(\bar{x}_1 \wedge x_3) &= 0\#\#\#\$0\#\#\#\$\#\#1\#\$\#\#1\#\$\$\$\$\$, \\ z_1 &= (01\#01\#01\#\$)^{10}. \end{aligned}$$

For f the threshold is 2.

$$\begin{aligned} r_2(f) &= ((1\#\#\#\$)^2(\#0\#\#\$)^2)\$\$\%)^2((0\#\#\#\$)^2(\#\#1\#\$)^2\$\$\$\$\%)^2\%\%\%\%, \\ z_2 &= ((01\#01\#01\#\$)^{10}\%)^{10}. \end{aligned}$$

The assignment strings for the assignment $a = 001$ are as follows.

$$\begin{aligned} s_0(a) &= 0\#0\#1\#, \\ s_1(a) &= (0\#0\#1\#\$01\#01\#01\#\$)^4, \\ s_2(a) &= ((0\#0\#1\#\$01\#01\#01\#\$)^4\%(01\#01\#01\#\$)^{10}\%)^4. \end{aligned}$$

Assignment a satisfies f and $r_2(f)$ is a subsequence of $s_2(a)$.

6. Discussion

We have shown that the class of shuffle ideals is not efficiently properly PAC learnable if RP $\neq$ NP, and is not efficiently improperly PAC learnable under certain cryptographic assumptions. On the other hand, even with classification noise, efficient proper PAC learning of shuffle ideals is possible under the uniform distribution. One technical question that remains is whether the results in Section 5 can be proved for an alphabet of constant size (independent of d.) Another is whether PAC learning shuffle ideals is as hard as PAC learning deterministic finite acceptors. Much remains to be understood about the learnability of subclasses of the regular languages.

Acknowledgments

A preliminary version of this paper appears in the ALT proceedings (Angluin et al., 2012). We thank the anonymous reviewers of ALT 2012 and JMLR for their helpful comments, and Dongqu Chen of Yale University for pointing out an error in the proof of Lemma 3. Improvements in the current version of the paper include a corrected proof of Lemma 3, reducing the minimum alphabet size for Theorem 7 from 3 to 2, and all of the results in Section 5.

The research of Dana Angluin and James Aspnes was supported by the National Science Foundation under grant CCF-0916389, that of Sarah Eisenstat by the T-Party Project (a joint research program between MIT and Quanta Computer Inc., Taiwan) and that of Aryeh Kontorovich by the Israel Science Foundation under grant No. 1141/12.

References

Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39(3):337–350, 1978. ISSN 0019-9958. doi: 10.1016/S0019-9958(78)90683-6.

Dana Angluin. Inference of reversible languages. *Journal of the ACM*, 29(3):741–765, July 1982. ISSN 0004-5411. doi: 10.1145/322326.322334.

Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, November 1987. ISSN 0890-5401. doi: 10.1016/0890-5401(87)90052-6.

Dana Angluin, James Aspnes, and Aryeh Kontorovich. On the learnability of shuffle ideals. In *Proceedings of the 23rd International Conference on Algorithmic Learning Theory*, ALT '12, pages 111–123, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-34105-2. doi: 10.1007/978-3-642-34106-9_12.

Nader H. Bshouty. Exact learning of formulas in parallel. *Machine Learning*, 26(1):25–41, January 1997. ISSN 0885-6125. doi: 10.1023/A:1007320031970.

Nader H. Bshouty, Jeffrey C. Jackson, and Christino Tamon. Exploring learnability between exact and PAC. *Journal of Computer and System Sciences*, 70(4):471–484, June 2005. ISSN 0022-0000. doi: 10.1016/j.jcss.2004.10.002.

Alexander Clark and Franck Thollard. PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5:473–497, December 2004. ISSN 1532-4435.

Corinna Cortes, Leonid (Aryeh) Kontorovich, and Mehryar Mohri. Learning languages with rational kernels. In *Proceedings of the 20th Annual Conference on Learning Theory*, COLT '07, pages 349–364, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-72925-9.

Colin de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9): 1332–1348, September 2005. ISSN 0031-3203. doi: 10.1016/j.patcog.2005.01.003.

Samuel Eilenberg and Saunders Mac Lane. On the groups of $H(\Pi, n)$, I. *Annals of Mathematics. Second Series*, 58:55–106, July 1953. ISSN 0003-486X.

Funda Ergün, S. Ravi Kumar, and Ronitt Rubinfeld. On learning bounded-width branching programs. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, COLT '95, pages 361–368, New York, NY, USA, 1995. ACM. ISBN 0-89791-723-5. doi: 10.1145/225298.225342.

E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

Yoshiyasu Ishigami and Sei'ichi Tani. VC-dimensions of finite automata and commutative finite automata with k letters and n states. *Discrete Applied Mathematics*, 74(3):229–240, May 1997. ISSN 0166-218X. doi: 10.1016/S0166-218X(96)00050-9.

Michael Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45 (6):983–1006, November 1998. ISSN 0004-5411. doi: 10.1145/293347.293351.

Michael Kearns and Leslie Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, January 1994. ISSN 0004-5411. doi: 10.1145/174644.174647.

Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA, 1994. ISBN 0-262-11193-4.

Ondřej Klíma and Libor Polák. Hierarchies of piecewise testable languages. In *Proceedings of the 12th International Conference on Developments in Language Theory*, DLT '08, pages 479–490, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-85779-2. doi: 10.1007/978-3-540-85780-8_38.

Leonid (Aryeh) Kontorovich and Boaz Nadler. Universal kernel-based learning with applications to regular languages. *Journal of Machine Learning Research*, 10:1095–1129, June 2009. ISSN 1532-4435.

Leonid (Aryeh) Kontorovich, Dana Ron, and Yoram Singer. A Markov model for the acquisition of morphological structure. *Technical Report CMU-CS-03-147*, June 2003.

Leonid (Aryeh) Kontorovich, Corinna Cortes, and Mehryar Mohri. Learning linearly separable languages. In *Proceedings of the 17th International Conference on Algorithmic Learning Theory*, ALT '06, pages 288–303, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-46649-5, 978-3-540-46649-9. doi: 10.1007/11894841_24.

Leonid (Aryeh) Kontorovich, Corinna Cortes, and Mehryar Mohri. Kernel methods for learning languages. *Theoretical Computer Science*, 405(3):223–236, October 2008. ISSN 0304-3975. doi: 10.1016/j.tcs.2008.06.037.

Kimmo Koskenniemi. Two-level model for morphological analysis. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI '83, pages 683–685, San Francisco, CA, USA, 1983. Morgan Kaufmann Publishers Inc.

M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics and Its Applications*. Addison-Wesley, 1983.

Mehryar Mohri. On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, 2(1):61–80, March 1996. ISSN 1351-3249. doi: 10.1017/S135132499600126X.

Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, June 1997. ISSN 0891-2017.

Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002. ISSN 0885-2308. doi: 10.1006/csla.2001.0184.

Mehryar Mohri, Pedro J. Moreno, and Eugene Weinstein. Efficient and robust music identification with weighted finite-state transducers. *IEEE Transactions on Audio, Speech & Language Processing*, 18(1):197–207, January 2010. ISSN 1063-6676. doi: 10.1109/TASL.2009.2023170.

José Oncina and Pedro García. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition*, volume 5 of *Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific Publishing, 1992.

Nick Palmer and Paul W. Goldberg. PAC-learnability of probabilistic deterministic finite state automata in terms of variation distance. *Theoretical Compututer Science*, 387(1):18–31, November 2007. ISSN 0304-3975. doi: 10.1016/j.tcs.2007.07.023.

Rajesh Parekh and Vasant G. Honavar. Learning DFA from simple examples. *Machine Learning*, 44(1-2):9–35, July 2001. ISSN 0885-6125. doi: 10.1023/A:1010822518073.

Gheorghe Păun, editor. *Mathematical Aspects of Natural and Formal Languages*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1994. ISBN 9-8102-1914-8.

Leonard Pitt and Manfred K. Warmuth. Prediction-preserving reducibility. *Journal of Computer and System Sciences*, 41(3):430–467, December 1990. ISSN 0022-0000. doi: 10.1016/0022-0000(90)90028-J.

Leonard Pitt and Manfred K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *Journal of the ACM*, 40(1):95–142, January 1993. ISSN 0004-5411. doi: 10.1145/138027.138042.

Owen Rambow, Srinivas Bangalore, Tahir Butt, Alexis Nasr, and Richard Sproat. Creating a finite-state parser with application semantics. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 2*, COLING '02, pages 1–5, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1071884.1071910.

Dana Ron, Yoram Singer, and Naftali Tishby. On the learnability and usage of acyclic probabilistic finite automata. *Journal of Computer and System Sciences*, 56(2):133–152, 1998. ISSN 0022-0000. doi: 10.1006/jcss.1997.1555.

Imre Simon. Piecewise testable events. In *Proceedings of the 2nd GI Conference on Automata Theory and Formal Languages*, pages 214–222, London, UK, 1975. Springer-Verlag. ISBN 3-540-07407-4.

Richard Sproat, William Gale, Chilin Shih, and Nancy Chang. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, 22(3):377–404, September 1996. ISSN 0891-2017.

Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984. ISSN 0001-0782.