On the Learnability of Shuffle Ideals

Dana Angluin*[1], James Aspnes*[1], and Aryeh Kontorovich**[2]

[1] Department of Computer Science
Yale University
New Haven, CT 06520.
[2] Department of Computer Science
Ben-Gurion University of the Negev
Beer Sheva, Israel 84105.

Abstract. Although PAC learning unrestricted regular languages is long known to be a very difficult problem, one might suppose the existence (and even an abundance) of natural efficiently learnable sub-families. When our literature search for a natural efficiently learnable regular family came up empty, we proposed the shuffle ideals as a prime candidate. A shuffle ideal generated by a string u is simply the collection of all strings containing u as a (discontiguous) subsequence. This fundamental language family is of theoretical interest in its own right and also provides the building blocks for other important language families. Somewhat surprisingly, we discovered that even a class as simple as the shuffle ideals is not properly PAC learnable, unless RP=NP. 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.

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 [10, 23, 25–28, 35, 40]. Thus, developing efficient formal-language learning techniques and understanding their limitations is of a broad and direct relevance in the digital realm.

Perhaps the most widely currently studied notion of learning is Valiant's PAC model [41], which allows for a clean, elegant theory while retaining a decent measure of empirical plausibility. Since PAC learnability is characterized by finite VC-dimension and the concept class of n-state Deterministic Finite-state Automata (DFA) has VC-dimension $\Theta(n \log n)$ [13], 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

* Supported in part by NSF grant CCF-0916389.
** This research was supported by the ISRAEL SCIENCE FOUNDATION (grant No. 1141/12).

works of Angluin [1], Gold [12] and Pitt and Warmuth [34], 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 [33] and Kearns and Valiant [16] 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 in [8, 30] to be learnable; see also the literature review therein. If the target automaton and sampling distribution are assumed to be "simple", efficient *probably exact* learning is possible [31]. When the learner is allowed to make *membership* queries, it follows from [3] that DFAs are learnable in this augmented PAC model.

The prevailing paradigm in formal 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 [2, 8, 29–31, 36] and, until recently, this appears to have been the only general-purpose technique available for learning finite automata.

More recently, Cortes et al. [9, 19, 20] 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 a follow-up work [21], 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.

Perhaps somewhat embarrassingly, there does not appear to be any known natural PAC-learnable family of regular languages. Let us qualify this statement to rule out the obvious objections. Many concept classes are known to be learnable over the boolean cube $\{0, 1\}^n$ — conjunctions, disjunctions, decision lists, etc. [17]. Another way to claim trivial results is by importing learning problems from continuous domains. For example, the concept class of axis-aligned rectangles in $\mathbb{R}^2$ is known to be PAC-learnable [17], so certainly these rectangles are also learnable over the rational plane $\mathbb{Q}^2$. Now we may identify $\mathbb{Q}^2$ with $\{0, 1\}^*$ via some bijection, thereby identifying rectangles over $\mathbb{Q}^2$ with languages $L \subset \{0, 1\}^*$ (we thank Kobbi Nissim for this example). It is a simple matter to construct a bijection $\phi : \mathbb{Q}^2 \to \{0, 1\}^*$ that maps rectangles to regular languages and vice versa. Observe, however, that we cannot a priori bound the size of the hypothesis automaton, since higher-precision rectangles will correspond to automata with more states. An even more basic reason to disqualify these examples is that it would be quite a stretch to call them "natural" families of regular languages.

What we mean by a PAC-learnable family of regular languages is, informally, the following. Fix some alphabet Σ. For $n \geq 1$, let $\mathcal{L}_n$ be a collection of regular languages, each of which is recognized by a DFA on n states or fewer. To avoid computational trivialities, let us rule out $|\mathcal{L}_n| = O(\mathrm{poly}(n))$ — this way, brute-force search is infeasible. Since, as we mentioned above and will see in more detail below, the information-theoretic aspects of the learning problem are well-understood, we focus here exclusively on the algorithmic ones. We say that $\mathcal{L} = \bigcup_n \mathcal{L}_n$ is **properly PAC learnable** if there is an algorithm that takes a labeled sample of size m and finds a consistent hypothesis in $\hat{L} \in \mathcal{L}_n$, in time $O(\mathrm{poly}(m, n))$. We say that $\mathcal{L}$ is **improperly PAC learnable** if there is an algorithm that takes a labeled sample of size m and finds a consistent hypothesis with description length $O(\mathrm{poly}(n))$, in time $O(\mathrm{poly}(m, n))$. A formal definition is given in Section 2.

Main results. Our main results concern the PAC-learnability of shuffle ideals. A shuffle ideal generated by a string u is simply the collection of all strings containing u as a (discontiguous) 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 in 1975 [39]. 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 [18, 24] and the references therein). On a more applied front, the shuffle ideals capture some rudimentary phenomena in human-language morphology [22]. In Section 3 we show that shuffle ideals of known length are exactly [5, 7] 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. Whether the shuffle ideals can be improperly PAC learned under general distributions remains an open question.

2 Preliminaries

Notation. 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. Define the binary relation $\sqsubseteq$ on Σ^* as follows: $u \sqsubseteq v$ holds if there is a witness $\boldsymbol{i} = (i_1 < i_2 < \ldots < i_{|u|})$ 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. We will write $I_{u \sqsubseteq v}$ to denote the position of the last symbol of u in its leftmost embedding in v (if the latter exists; otherwise, $I_{u \sqsubseteq v} = \infty$).

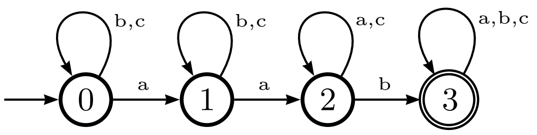

Fig. 1. The canonical DFA for recognizing the shuffle ideal of $u = aab$ over $\Sigma = \{a, b, c\}$, which accepts precisely those strings that contain u as a subsequence.

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

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

(an example is given in Figure 1). The term *shuffle ideal* comes from algebra [24, 32] and dates back to [11].

We use the standard $O(\cdot)$, $o(\cdot)$ notation to denote orders of magnitude. The following simple observation will be useful in the sequel.

Lemma 1. *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 $O(|x|)$. □

Learnability. We assume a familiarity with the basics of the PAC learning model [17]. 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 $\epsilon, \delta > 0$ 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

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

moreover, we require that both m_0 and $\mathcal{L}$'s runtime be at most polynomial in $\epsilon^{-1}, \delta^{-1}$. The learning is said to be *proper* if $\mathcal{H} = \mathcal{C}$ and *improper* otherwise.

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

$$O\left(\frac{n \log |\Sigma| - \log \delta}{m}\right). \tag{1}$$

Hence, the problem of properly PAC-learning shuffle ideals has been reduced to finding one that is consistent with a given sample. (This justifies our informal problem statement in the introduction, where the requirements are purely algorithmic and no mention of ϵ, δ is made.) This will turn out to be computationally hard under adversarial distributions (Theorem 4), but feasible under the uniform one (Theorem 3). Actually, our positive result is somewhat stronger: since we show learnability in the statistical query (SQ) model of Kearns [15], this implies a noise-tolerant PAC-result.

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 $\mathcal{X} = \Sigma^{n}$ endowed with the uniform distribution, which assigns a weight of $|\Sigma|^{-n}$ to each element of $\mathcal{X}$. Our learning algorithm is most naturally expressed in the language of *statistical queries* [15,17]. 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 $u \in \Sigma^{\leq n}$ and $a \in \Sigma$, we define the query $\chi_{u,a}(\cdot,\cdot)$ by

$$\chi_{u,a}(x,y) = \begin{cases} 0, & u \not\sqsubseteq x \\ \mathbb{1}_{\{\sigma=a\}} - \mathbb{1}_{\{\sigma\neq a\}}/(s-1), & u \sqsubseteq x,\ y=+1 \\ \mathbb{1}_{\{\sigma\neq a\}}/(s-1) - \mathbb{1}_{\{\sigma=a\}}, & u \sqsubseteq x,\ y=-1 \end{cases}, \tag{2}$$

where σ is the symbol in x following the leftmost embedding of u (formally, $\sigma = x_{I_{x\sqsubseteq u}+1}$) 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 1) 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 2.

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

where

$$P(L,n,s) = \binom{n-1}{L-1}\left(\frac{1}{s}\right)^{L-1}\left(1-\frac{1}{s}\right)^{n-L}. \tag{3}$$

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, i.e., 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:

$$T \sim \text{Binom}(n, 1/s); \tag{4}$$

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 $\ell < L$. Let $I_\ell = I_{u \sqsubseteq X}$ be the position of u_ℓ in the leftmost embedding of $u_1 \ldots u_\ell$ in X (0 if $\ell = 0$), or $n-1$ if $u_1 \ldots u_\ell$ is not a subsequence of X. Then $I_\ell + 1$ is the position of σ as defined in (2), or n if $u_1 \ldots u_\ell \not\sqsubseteq X_1 \ldots X_{n-1}$.

Define T_ℓ to be the number of symbols of a leftmost embedding of u' in X excluding $X_{I_\ell+1}$:

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

Like T, T_ℓ also has a binomial distribution, but now

$$T_\ell \sim \text{Binom}(n-1, 1/s). \tag{5}$$

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'.

An important fact is that T_ℓ is independent of $X_{I_\ell+1}$. 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' is still an independent $1/s$ whether $X_{I_\ell+1}$ consumes a symbol of u' or not. So the distribution of T_ℓ is not affected.

We now compute $\mathbf{E}\chi_{u,a}$ by averaging over all choices of T_ℓ. If $T_\ell < \ell$, then $u_1 \ldots u_\ell \not\sqsubseteq X_1 \ldots X_{n-1}$ and $\chi_{u,a} = 0$. If $\ell \le T_\ell \le L-2$, then X is a negative example. Each symbol in Σ contributes 1 to the mean with probability $1/s$ and $-\frac{1}{s-1}$ with probability $\frac{s-1}{s}$; the net contribution is 0. Similarly, if $T_\ell \ge L$, X is a positive example, and the probability-$(1/s)$ gain of 1 is exactly offset by the probability-$\left(\frac{s-1}{s}\right)$ loss of $\frac{1}{s-1}$.

This leaves the case $T_\ell = L-1$. Now X is positive if and only if $X_{I_\ell+1} = \bar{u}_{\ell+1}$, which occurs if $\sigma = \bar{u}_{\ell+1}$. So the conditional expectation is $1 \cdot \Pr[\sigma = \bar{u}_{\ell+1}] + \frac{1}{s-1} \cdot \Pr[\sigma \ne \bar{u}_{\ell+1}] = \frac{1}{s} + \frac{1}{s-1} \cdot \frac{s-1}{s} = 2/s$. For $a \ne \bar{u}_{\ell+1}$, 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 $a \ne \bar{u}_{\ell+1}$ induce same expectation by symmetry.

Since the only case that produces a nonzero conditional expectation is $T_\ell = L-1$, we have

$$\mathbf{E}\chi_{u,\bar{u}_{\ell+1}} = +\frac{2}{s}\Pr[T_\ell = L-1], \tag{6}$$

and for each $a \ne \bar{u}_{\ell+1}$,

$$\mathbf{E}\chi_{u,a} = -\frac{2}{s(s-1)}\Pr[T_\ell = L-1]. \tag{7}$$

The claim follows since $T_\ell \sim \mathrm{Binom}(n-1, 1/s)$ by (5). □

3.2 Specifying the query tolerance τ

The analysis in Lemma 2 suggests that inferring $\bar{u} \in \Sigma^L$ amounts to distinguishing the two possible values of $\mathbf{E}\chi_{u,a}$. If we set the query tolerance to half the larger value

$$\tau = \frac{1}{s}\Pr[T_\ell = L-1] \tag{8}$$

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

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

In the above SQ algorithm there is no need for a precision parameter ϵ because the learning is *exact*, that is, $\epsilon = 0$. 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 ($\epsilon > 0$), we can modify the above algorithm to use a polynomially bounded tolerance.

Theorem 2. *When the length L of the target string $\bar{u}$ is known, $\bar{u}$ is approximately identifiable to within $\epsilon > 0$ with $O(Ls)$ statistical queries at tolerance $\tau = \epsilon/(3sn)$.*

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

Otherwise, $\Pr[\xi = 1]$ is between $\epsilon/3$ and $1 - \epsilon/3$, and the first SQ algorithm is used. We now show that $P(L, n, s) \geq \epsilon/(3n)$, 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 $Q(L, n, s) \geq \epsilon/(3L)$ and $P(L, n, s) \geq \epsilon/(3n)$. 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 $P(L, n, s) \geq Q(L, n, s) \geq \epsilon/(3n)$. □

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 3. *The concept class $\mathcal{C} = \left\{ Ш(u) : u \in \Sigma^{\leq n} \right\}$ 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 $\tau = \epsilon/(3sn)$, as described in [15]. According to [15, Theorem 1],

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

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

□

Remark 1. Our learning algorithm and analysis are rather strongly tied to the uniform distribution. If this assumption is omitted, it might now happen that $\Pr[T_\ell = m - 1]$ is small even though positive and negative examples are mostly balanced, or there might be intractable correlations between σ and T_ℓ. 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

Our hardness result will follow the standard paradigm, exemplified in [17]. We will 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≠NP, there is no polynomial-time algorithm that properly learns shuffle ideals.

Theorem 4. *Given two disjoint sets of strings $S, T \subset \Sigma^*$, the problem of determining whether there exists a string w such that $w \sqsubseteq x$ for each $x \in S$ and $w \not\sqsubseteq x$ for each $x \in T$ is NP-complete.*

Proof. To see that this 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 1).

To see that this problem is complete in NP, we reduce satisfiability of 3-CNF formulas to this question. Given a formula ϕ containing n clauses C_i, where each clause contains three literals $\ell_{i,1}$, $\ell_{i,2}$ and $\ell_{i,3}$, the question of whether ϕ is satisfiable is equivalent to the question of whether we can select exactly one literal from each clause in such a way that no two selected literals are complements of each other.

The heart of the construction is a three-way choice of one part of a subsequence for each clause of the formula. Consider the strings $x_1 = aba$ and $x_2 = baab$. The strings that are subsequences of both of these strings are precisely

$$\{\lambda, a, b, aa, ab, ba\}.$$

Thus if we were to specify positive strings x_1 and x_2, and negative strings a and b, there are exactly three strings that are subsequences of both the positive strings and not subsequences of either of the negative strings, namely, $\{aa, ab, ba\}$. We use n copies of this three-way choice to represent the choice of one literal from each of the n clauses.

We define S to contain the two positive strings:

$$u_1 = (x_1 d)^n$$
$$u_2 = (x_2 d)^n$$

where the symbol d acts to delimit the region of each string corresponding to each clause.

Our first group of negative strings, T_1, contains the n strings obtained from u_1 by deleting one occurrence of d. A string w that is a subsequence of u_1 and not a subsequence of any string in T_1 must have exactly n occurrences of

d. The occurrences of d divide w into regions corresponding to the successive occurrences of x_1 in u_1 and x_2 in u_2.

Our second group of negative strings, T_2, contains the $2n$ strings obtained from u_1 by selecting a region i and replacing the x_1 in that region of u_1 by a or b. We precisely characterize the set of strings w that are subsequences of u_1 and u_2 but not of any string in T_1 or T_2 as the strings described by the regular expression $((y_1 + y_2 + y_3)d)^n$, where $y_1 = aa$, $y_2 = ab$, and $y_3 = ba$. In region i we associate the choice of string y_r with choosing the literal $\ell_{i,r}$.

Finally, our third group of negative strings, T_3, contains a string for each pair of complementary literals (say $\ell_{i,r}$ and $\ell_{j,s}$) obtained from u_1 as follows. In region i we substitute y_r for x_1, and in region j we substitute y_s for x_1. This negative string means that a consistent string w cannot make a choice of strings corresponding to the complementary literals $\ell_{i,r}$ and $\ell_{j,s}$.

Then there is a string w that is a subsequence of every string in the positive set S and of no string in the negative set $T = T_1 \cup T_2 \cup T_3$ if and only if the original formula ϕ is satisfiable, concluding the NP-completeness proof. $\square$

5 The Difficulty of Learning Unions of Shuffle Ideals

In this section we note that the problem of learning a monotone DNF formula is efficiently reducible to the problem of learning a union of shuffle ideals. Let ϕ be a monotone DNF formula over variables $\{x_i\}$ for $i = 1, \ldots, n$. We consider an ordered alphabet $\{x_1, \ldots, x_n\}$ and a union h of shuffle ideals obtained from ϕ as follows. Each term, e.g., $x_6x_{14}x_{22}$, of ϕ is mapped to a shuffle ideal consisting of the symbols (in order) corresponding to the variables in the term, e.g., $\text{Ш}(x_6x_{14}x_{22})$. Then h is the union of the shuffle ideals obtained in this way.

If we map each assignment to the variables $\{x_i\}$ to the substring of

$$x_1x_2 \cdots x_n$$

obtained by omitting the symbols corresponding to variables assigned 0, then the assignments satisfying ϕ are precisely the substrings of $x_1x_2 \cdots x_n$ that are in the union of shuffle ideals h. Thus an efficient method of learning unions of shuffle ideals would yield an efficient method of learning for monotone DNF formulas, which so far is only known in special cases [4, 6, 14, 37, 38].

6 Discussion

We have shown that even a family of regular languages as simple as the shuffle ideals is not efficiently properly PAC learnable if RP≠NP. Thus, the search for a nontrivial (in the sense described in the introduction) properly PAC-learnable family of languages continues. On the other hand, even with classification noise, efficient proper PAC learning of shuffle ideals is possible under the uniform distribution. The major unresolved question is whether it is possible to *improperly* learn shuffle ideals under general distributions; this is the subject of ongoing research. Also open is the question of whether the alphabet size in Theorem 4 can be reduced to 2.

Acknowledgments

We thank Sarah Eisenstat for the construction reducing the alphabet size to 3 in the proof of Theorem 4 and the anonymous referees for their helpful comments.

References

1. Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 3(39):337–350, 1978.
2. Dana Angluin. Inference of reversible languages. *Journal of the ACM (JACM)*, 3(29):741–765, 1982.
3. Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.
4. Dana Angluin and Donna K. Slonim. Randomly fallible teachers: Learning monotone DNF with an incomplete membership oracle. *Machine Learning*, 14(1):7–26, 1994.
5. Nader H. Bshouty. Exact learning of formulas in parallel. *Machine Learning*, 26(1):25–41, 1997.
6. Nader H. Bshouty and Nadav Eiron. Learning monotone DNF from a teacher that almost does not answer membership queries. *Journal of Machine Learning Research*, 3:49–57, 2002.
7. Nader H. Bshouty, Jeffrey C. Jackson, and Christino Tamon. Exploring learnability between exact and PAC. *J. Comput. Syst. Sci.*, 70(4):471–484, 2005.
8. Alexander Clark and Franck Thollard. Pac-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research (JMLR)*, 5:473–497, 2004.
9. Corinna Cortes, Leonid (Aryeh) Kontorovich, and Mehryar Mohri. Learning languages with rational kernels. In *COLT*, pages 349–364, 2007.
10. Colin de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38:1332–1348, 2005.
11. Samuel Eilenberg and Saunders Mac Lane. On the groups of $H(\Pi, n)$. I. *Ann. of Math. (2)*, 58:55–106, 1953.
12. E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 3(37):302–420, 1978.
13. Yoshiyasu Ishigami and Seiichi Tani. Vc-dimensions of finite automata and commutative finite automata with k letters and n states. *Discrete Applied Mathematics*, 74(2):123 – 134, 1997.
14. Jeffrey C. Jackson, Homin K. Lee, Rocco A. Servedio, and Andrew Wan. Learning random monotone DNF. *Discrete Applied Mathematics*, 159(5):259–271, 2011.
15. Michael Kearns. Efficient noise-tolerant learning from statistical queries. *J. ACM*, 45(6):983–1006, November 1998.
16. Michael J. Kearns and Leslie G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.
17. Micheal Kearns and Umesh Vazirani. *An Introduction to Computational Learning Theory.* The MIT Press, 1997.
18. Ondrej Klíma and Libor Polák. Hierarchies of piecewise testable languages. *Int. J. Found. Comput. Sci.*, 21(4):517–533, 2010.
19. Leonid (Aryeh) Kontorovich, Corinna Cortes, and Mehryar Mohri. Learning linearly separable languages. In *ALT*, pages 288–303, 2006.
20. Leonid (Aryeh) Kontorovich, Corinna Cortes, and Mehryar Mohri. Kernel methods for learning languages. *Theor. Comput. Sci.*, 405(3):223–236, 2008.

21. Leonid (Aryeh) Kontorovich and Boaz Nadler. Universal Kernel-Based Learning with Applications to Regular Languages. *Journal of Machine Learning Research*, 10:997–1031, 2009.
22. Leonid (Aryeh) Kontorovich, Dana Ron, and Yoram Singer. A Markov Model for the Acquisition of Morphological Structure. *Technical Report CMU-CS-03-147*, 2003.
23. Kimmo Koskenniemi. Two-level model for morphological analysis. In *IJCAI*, pages 683–685, 1983.
24. M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics and Its Applications*. Addison-Wesley, 1983.
25. Mehryar Mohri. On some applications of finite-state automata theory to natural language processing. *Nat. Lang. Eng.*, 2:61–80, March 1996.
26. Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.
27. Mehryar Mohri, Pedro 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, 2010.
28. Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.
29. José Oncina and Pedro García. *Identifying regular languages in polynomial time*, pages 49–61. World Scientific Publishing, 1992. Advances in Structural and Syntactic Pattern Recognition,.
30. Nick Palmer and Paul W. Goldberg. PAC-learnability of probabilistic deterministic finite state automata in terms of variation distance. *Theor. Comput. Sci.*, 387(1):18–31, 2007.
31. Rajesh Parekh and Vasant G. Honavar. Learning DFA from simple examples. *Mach. Learn.*, 44(1-2):9–35, 2001.
32. Gheorghe Păun. *Mathematical Aspects of Natural and Formal Languages.* World Scientific Publishing, 1994.
33. Leonard Pitt and Manfred Warmuth. Prediction-preserving reducibility. *Journal of Computer and System Sciences*, 41(3):430–467, 1990.
34. Leonard Pitt and Manfred Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *Journal of the Association for Computing Machinery*, 40(1):95–142, 1993.
35. Owen Rambow, Srinivas Bangalore, Tahir Butt, Alexis Nasr, and Richard Sproat. Creating a finite-state parser with application semantics. In *COLING*, 2002.
36. 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.
37. Linda Sellie. Learning random monotone DNF under the uniform distribution. In *COLT*, pages 181–192, 2008.
38. Rocco A. Servedio. On learning monotone DNF under product distributions. *Inf. Comput.*, 193(1):57–74, 2004.
39. Imre Simon. Piecewise testable events. In *Automata Theory and Formal Languages*, pages 214–222, 1975.
40. Richard Sproat, Chilin Shih, William Gale, and Nancy Chang. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, 22(3):377–404, 1996.
41. Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.