Learning Linearly Separable Languages

Leonid Kontorovich
May 13, 2004
CMU-CALD-04-105

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

For a finite alphabet $\mathcal{A}$, we define a class of embeddings of $\mathcal{A}^*$ into an infinite-dimensional feature space $\mathcal{X}$ and show that its finitely supported hyperplanes define regular languages. This suggests a general strategy for learning regular languages from positive and negative examples. We apply this strategy to the piecewise testable languages, presenting an embedding under which these are precisely the linearly separable ones, and thus are efficiently learnable.

Most of this work was done while the author was visiting the Hebrew University, Jerusalem, Israel. This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. The research at CMU was supported in part by NSF ITR grant IIS-0205456. This publication only reflects the author's views.

Keywords: regular language, piecewise testable, learning, kernel, strings

1 Introduction

The problem of learning regular languages from positive and negative examples has occupied researchers for some time. Specifically, we have a finite alphabet $\mathcal{A}$ and the teacher has in mind some target language $L \subset \mathcal{A}^*$. The teacher shows us two sets of strings $S^+, S^- \subset \mathcal{A}^*$, where $S^+ \subset L$ are the *positive* examples and $S^- \not\subset L$ are the *negative* examples. Our task is to infer L from the sample $S = S^+ \cup S^-$.

Different learning models have been studied, including *language identification in the limit* [7], building the *smallest consistent automaton* [1, 8], and the PAC model [10, 14, 15] and strong negative results have been obtained. Rather than provide a comprehensive survey of the literature on the subject, let us quote Rivest and Schapire [16]: "The problem of learning an automaton by passively observing [a labeled sample] is now well established to be a hard computational problem" and refer the reader to the survey therein.

Positive results have been obtained in a few special cases, most notably Trakhtenbrot and Barzdin's learning a DFA from a *complete labeled set* [20], Angluin's learning *reversible languages* [3] and Ron et al.'s learning *acyclic probabilistic automata* [17] (the latter are a class of probabilistic automata containing a class of ordinary DFAs as degenerate cases). A number of common techniques emerge from these positive results. One generally builds a prefix-tree acceptor consistent with the sample and performs some sort of merging of the states. The principal difficulty lies in finding the merging criterion appropriate for the given problem and in proving correctness and time and sample efficiency of the algorithm.

In this work, we depart from the established paradigm. Instead of building an automaton from the sample S, we will embed the strings in $\mathcal{A}^*$ into an inner product space and learn a hyperplane that separates S^+ and S^-. We will define a class of embeddings in which linear separators correspond to regular languages. We will provide a natural embedding for a particular class of regular languages (the piecewise testable ones), as well as an efficient kernel for this embedding, which will immediately yield a learning algorithm for this class.

This paper is organized as follows. In Sect. 2 we define a general way to map strings in $\mathcal{A}^*$ to vectors in an inner product space and prove correspondences between linear separators and regular languages. In Sect. 3 we outline a general strategy for learning families of regular languages. In Sect. 4 we apply this strategy to the piecewise testable languages. Finally, in Sections 5 and 6 we summarize our results and suggest directions for future research.

A note on the notation: boldfaced symbols (such as $\boldsymbol{\varphi}$ or $\mathbf{w}$) will denote vectors or vector-valued functions; their components will be unboldfaced and with subscripts (such as φ_i or w_i). For strings u over some alphabet $\mathcal{A}$, we write $|u|$ to denote u's length and $u[i]$ to denote the ith character of u.

2 Regular Locally Finite Covers

2.1 Preliminaries

Fix an alphabet $\mathcal{A}$ and consider a family of sets $U_i \subset \mathcal{A}^*$, for $i = 1, 2, \ldots$, such that each $s \in \mathcal{A}^*$ lies in at least one and at most finitely many U_i:

$$1 \leq \sum_i \varphi_i(s) < \infty, \tag{1}$$

where φ_i is the characteristic function of U_i on $\mathcal{A}^*$:

$$\varphi_i(s) = \begin{cases} 1, & s \in U_i \\ 0, & s \notin U_i \end{cases} .$$

Any such family $\{U_i\}$ is called a *locally finite cover* of $\mathcal{A}^*$. If, in addition, each U_i is a regular set, $\{U_i\}$ is called a Regular Locally Finite Cover (RLFC). As a technicality, we require that $\emptyset$ always belong to an RLFC.

Any locally finite cover $\{U_i\}$ defines a kernel K over $\mathcal{A}^*$:

$$K(s,t) = \sum_i \varphi_i(s)\varphi_i(t) \tag{2}$$

whose finiteness, symmetry and positive definiteness follow trivially from the construction. Intuitively, $K(s,t)$ counts the number of cover elements U_i to which both s and t belong.

We may view $\boldsymbol{\varphi}(s)$ as an infinite-dimensional vector in the space $\mathcal{X} = \mathbb{R}^\infty$, in which case we can write $K(s,t) = \langle \boldsymbol{\varphi}(s), \boldsymbol{\varphi}(t) \rangle$; we will call $\boldsymbol{\varphi}$ an *RLFC-induced* embedding. We will say that an (oriented) hyperplane defined by $\mathbf{w} \in \mathcal{X}$ is *finitely supported* if $\mathbf{w}$ has finite support (finitely many nonzero coordinates) in $\mathcal{X}$.

Note that any $\mathbf{w} \in \mathcal{X}$ defines a language $L(\mathbf{w})$ in the following way:

$$L(\mathbf{w}) = \{s \in \mathcal{A}^* : \langle \mathbf{w}, \boldsymbol{\varphi}(s) \rangle > 0\}. \tag{3}$$

The language L is said to be linearly separable (with respect to a given embedding $\boldsymbol{\varphi}$) if there exists a finitely supported hyperplane defined by $\mathbf{w} \in \mathcal{X}$ such that $L = L(\mathbf{w})$.

2.2 Main result

The main claim of this section is that any finitely supported hyperplane under an RLFC embedding defines a regular language. (The converse is clearly false – for a given RLFC, it is not necessarily the case that every regular language is given by some hyperplane.[1])

Let us state the claim in its full formality:

Theorem 2.1. *Given an RLFC-induced $\boldsymbol{\varphi} : \mathcal{A}^* \to \mathcal{X}$ and a finitely supported $\mathbf{w} \in \mathcal{X}$, the language $L(\mathbf{w}) = \{s \in \mathcal{A}^* : \langle \mathbf{w}, \boldsymbol{\varphi}(s) \rangle > 0\}$ is regular.*

Proof. Define $f : \mathcal{A}^* \to \mathbb{R}$ by

$$f(s) \equiv \langle \mathbf{w}, \boldsymbol{\varphi}(s) \rangle \tag{4}$$

$$= \sum_{i=1}^{N} w_i \varphi_i(s), \tag{5}$$

where the constants $\{w_i\}$ and N are independent of s (the finiteness of N follows from the assumption that $\mathbf{w}$ is finitely supported).

Thus, f is a finite linear combination of binary features φ_i, with real coefficients w_i. Observe that $f(\cdot)$ can only take on finitely many real values. Let $\{r_k\}_{k=1}^K$ be the range of f. Define $L_{r_k} \subset \mathcal{A}^*$ by

$$L_{r_k} = f^{-1}(r_k). \tag{6}$$

[1] A simple counterexample is furnished the RLFC $\{\emptyset, U, \mathcal{A}^* \setminus U\}$ for some regular language U. For this RLFC, U is linearly separable but no other nonempty regular language is.

For any subset of features indexed by $I \subset 2^{\{1,2,\ldots,N\}}$, call it r_k-acceptable if $\sum_{i\in I} w_i = r_k$. Each such r_k-acceptable set corresponds to a set of strings $L_I \subset \mathcal{A}^*$ such that

$$L_I = \left(\bigcap_{i\in I} \varphi_i^{-1}(1)\right) \setminus \left(\bigcup_{i\in\{1,\ldots,N\}\setminus I} \varphi_i^{-1}(1)\right) \tag{7}$$

$$= \left(\bigcup_{i\in I} U_i\right) \setminus \left(\bigcap_{i\in\{1,\ldots,N\}\setminus I} U_i\right) \tag{8}$$

so L_I is regular because each U_i is regular (by definition of RLFC). Then each L_{r_k} is the union of finitely many r_k-acceptable L_I's, and L is the union of the L_{r_k} for positive r_k. □

2.3 Representer theorem

Given a finite set of strings $S = \{s_j\}_{j=1}^m \subset \mathcal{A}^*$ and a "weights" vector $\boldsymbol{\alpha} \in \mathbb{R}^m$, these define the language $L(S, \boldsymbol{\alpha}) = \{s \in \mathcal{A}^* : \sum_{j=1}^m \alpha_j K(s_j, s) > 0\}$, which is seen to be regular by taking the finitely supported hyperplane to be defined by $\mathbf{w} = \sum_j \alpha_j \boldsymbol{\varphi}(s_j)$ and applying Thm. 2.1. Let us call the languages for which such a pair $(S, \boldsymbol{\alpha})$ exists *strongly linearly separable*. We have just demonstrated that any strongly linearly separable language is linearly separable. The converse is similarly straightforward:

Theorem 2.2. *Given an RLFC-induced $\boldsymbol{\varphi} : \mathcal{A}^* \to \mathcal{X}$ and a finitely supported $\mathbf{w} \in \mathcal{X}$, let $L(\mathbf{w}) = \{s \in \mathcal{A}^* : \langle \mathbf{w}, \boldsymbol{\varphi}(s)\rangle > 0\}$. Then there exist $\{s_j\}_{j=1}^m \subset \mathcal{A}^*$ and $\boldsymbol{\alpha} \in \mathbb{R}^m$ such that $L(S, \boldsymbol{\alpha}) \equiv \{s \in \mathcal{A}^* : \sum_{j=1}^m \alpha_j K(s_j, s) > 0\} = L(\mathbf{w})$.*

Proof. Let N be the largest nonzero coordinate of $\mathbf{w}$. Then $\boldsymbol{\varphi}$ is effectively a mapping from $\mathcal{A}^*$ to $\mathbb{R}^N$, since the higher coordinates play no role in determining whether a string belongs to L; let us formally denote this by $\widehat{\boldsymbol{\varphi}} : \mathcal{A}^* \to \mathbb{R}^N$. We have that $\widehat{\boldsymbol{\varphi}}(L)$ and $\widehat{\boldsymbol{\varphi}}(\mathcal{A}^* \setminus L)$ are separated by some hyperplane in $\mathbb{R}^N$; let $\mathbf{w}'$ be the maximum-margin separating hyperplane[2]. Let $S \subset \mathcal{A}^*$ be such that $\widehat{\boldsymbol{\varphi}}(S)$ is the set of support vectors for $\mathbf{w}'$. If S is finite then we are done, otherwise choose a finite $S' \subset S$ such that $\mathrm{span}(\widehat{\boldsymbol{\varphi}}(S')) = \mathrm{span}(\widehat{\boldsymbol{\varphi}}(S))$, where $\mathrm{span}(\cdot)$ is the linear span of a set of vectors. □

This last result shows that for any linearly separable language there exists a finite sample from which it can be inferred.

Theorem 2.1 gives a representation of regular languages in terms of certain sets in $\mathcal{X}$. Although we present a construction for converting this representation to a more familiar one (such as an FSA), our construction is not necessarily efficient. Indeed, for any r_k there may be exponentially many r_k-acceptable L_I's. This underscores the unique feature of our methodology: our goal is to efficiently learn regular *languages* (in some representation) – not necessarily automata.

2.4 Further Characterization

It is natural to ask what property of finitely supported hyperplanes is responsible for their inducing regular languages. In fact, Thm. 2.1 is readily generalized:

[2] Although at least one of the two sets $\widehat{\boldsymbol{\varphi}}(L)$ and $\widehat{\boldsymbol{\varphi}}(\mathcal{A}^* \setminus L)$ is infinite, the fact that $\boldsymbol{\varphi}$ is a coordinatewise integer-valued function implies that the margin will be nonzero.

Theorem 2.3. *Let $f : \mathcal{A}^* \to \mathbb{R}$ be of the form $f(s) = g(\boldsymbol{\varphi}(s)) = g(\varphi_1(s), \varphi_2(s), \ldots, \varphi_N(s))$ (that is, the value of f depends on a fixed finite number of the components of $\boldsymbol{\varphi}$). Then for any $r \in \mathbb{R}$, the language $L = \{s \in \mathcal{A}^* : f(x) = r\}$ is regular.*

Proof. Since f is a function of finitely many binary variables, its range is finite. From here, the proof proceeds exactly as the proof of Thm. 2.1, with identical definitions for $\{r_k\}$, L_{r_k}, etc. □

Corollary 2.4. *Let $f : \mathcal{A}^* \to \mathbb{R}$ satisfy the condition of Thm. 2.3. Then for any $r \in \mathbb{R}$, the languages $L_1 = \{s \in \mathcal{A}^* : f(s) > r\}$ and $L_2 = \{s \in \mathcal{A}^* : f(s) < r\}$ are regular.*

Remark 2.5. The theorem need not hold if f is allowed to depend on arbitrarily many components of $\boldsymbol{\varphi}$ for then f might take on infinitely many values. Even if we restrict the range of f to be finite, the number of r_k-acceptable languages L_I might be infinite for some r_k, and then L_{r_k} would no longer be a finite union of regular languages.

For example, let $\mathcal{A} = \{\mathsf{a}\}$ and let U_i be the singleton $\{\mathsf{a}^i\}$, for $i = 0, 1, \ldots$. The $\{U_i\}$ clearly form an RLFC. Define $f : \mathcal{A}^* \to \{0, 1\}$ by $f(s) = \begin{cases} 1, & |s| \text{ is prime} \\ 0, & \text{else} \end{cases}$. Then f has finite range, but $f^{-1}(1)$ is not a regular language.

3 General Learning Strategy

The definitions and results of the previous section suggest a natural paradigm for learning regular languages from labeled examples. Let $\mathcal{F}$ be a family of regular languages that we would like to learn. Our first step would be to look for a suitable RLFC for $\mathcal{F}$. Namely, the RLFC must be such that every $L \in \mathcal{F}$ is linearly separable under the embedding $\boldsymbol{\varphi}$ induced by the RLFC. If we succeed in finding an appropriate RLFC, we would need to come up with an efficient algorithm for computing the associated kernel (2). This might not be trivial, since any string might belong to exponentially many cover elements and the sum in (2) might be intractable.

If we manage to complete both of these stages, we shall have a learning algorithm for $\mathcal{F}$. Given a finite sample of strings, we can run kernelized SVM [4] to obtain a separating hyperplane, which in turn induces a regular language (Thm. 2.1). Our choice of a good RLFC guarantees that the sample will be separable, and the standard generalization error bounds for SVMs will hold [21, 9].[3]

We shall presently apply this strategy to a particular family of regular languages – the piecewise testable ones.

4 Piecewise Testable Languages

4.1 Definitions

Piecewise testable languages were defined in Imre Simon's PhD thesis and in [18].

For $s \in \mathcal{A}^n$, define the *shuffle ideal* of s to be the set of all strings containing s as a subsequence:

$$\text{shuff}(s) = \mathcal{A}^* s[1] \mathcal{A}^* \ldots \mathcal{A}^* s[n] \mathcal{A}^* \tag{9}$$

(define $\text{shuff}(\varepsilon) = \mathcal{A}^*$).

[3] As a technicality, we note that the theory of reproducing kernels requires our inner product space to be a Hilbert space, i.e., complete in the norm induced by the inner product. Our space $\mathcal{X}$ will not necessarily be complete in the inner product induced by an RLFC. However, since any string can only belong to finitely many cover elements, for any finite sample we are effectively working in a finite-dimensional space, which is a Hilbert space.

A language is piecewise testable iff it is a finite boolean combination of shuffle ideals[4]. For example,

$$L = ((\mathcal{A}^*\mathsf{a}\mathcal{A}^*) \cup (\mathcal{A}^*\mathsf{b}\mathcal{A}^*)) \cap \overline{((\mathcal{A}^*\mathsf{a}\mathcal{A}^*) \cap (\mathcal{A}^*\mathsf{b}\mathcal{A}^*))}$$

is a piecewise testable language over $\mathcal{A} = \{\mathsf{a}, \mathsf{b}\}$.

It might not at first be obvious how, given a regular language in some representation, to determine whether it is piecewise testable; Simon, however, showed this problem to be efficiently decidable. The piecewise testable languages have received a fair amount of attention in the formal-language community, as a major subclass of the dot-depth one hierarchy [19, 6].

Our key observation is that the shuffle ideals – languages of the form $U = \mathcal{A}^*a_1\mathcal{A}^*a_2\mathcal{A}^* \ldots \mathcal{A}^*a_k\mathcal{A}^*$, $k \geq 0, a_i \in \mathcal{A}$ – form a Regular Locally Finite cover of $\mathcal{A}^*$ (together with $\emptyset$). Indeed, every shuffle ideal is a regular set, every string $s \in \mathcal{A}^*$ will belong to a shuffle ideal (e.g., $\mathrm{shuff}(s)$), and a string s can only belong to finitely many shuffle ideals (namely, if $|s| < |t|$ then $s \notin \mathrm{shuff}(t)$).

Thus the shuffle ideals are a natural candidate for an RLFC for the piecewise testable languages; in the next section we verify that these languages are indeed linearly separable under this RLFC.

4.2 Piecewise Testable Languages are Linearly Separable

Remark 4.1. In this section, *linear separability* will always mean with respect to the embedding induced by the shuffle-ideals RLFC.

If L is a regular language, let $A(L)$ denote the minimum DFSA recognizing L. Let $\Gamma(L)$ denote the directed graph underlying the automaton. We say that Γ is *acyclic* if it has no cycles except for possible self-loops. Note than in an acyclic graph, there is a natural partial order on the nodes: we write $p \prec q$ if there is a directed path from p to q in Γ. For $\mathcal{A}' \subset \mathcal{A}$, let $\Gamma(\mathcal{A}')$ denote the subgraph induced by the nodes of Γ and the edges labeled by members of $\mathcal{A}'$. Simon [18] characterizes the piecewise testable languages by their graphs:

Theorem 4.2. *Let L be a regular language over $\mathcal{A}$. Then L is piecewise testable iff*
(i) Γ is acyclic
(ii) for any $\mathcal{A}' \subset \mathcal{A}$, each connected component of $\Gamma(\mathcal{A}')$ has a unique maximal state.

Our major result is that

Theorem 4.3. *Piecewise testable languages are linearly separable.*

Proof. Let $A = (Q, \mathcal{A}, \delta, q_0, F)$ be the minimal automaton recognizing some piecewise testable language L. Let us say that $s \in \mathcal{A}^*$ is a *path* if in the sequence of states $q_0, q_1 = \delta(q_0, s[1]), q_2 = \delta(q_1, s[2]), \ldots, q_{|s|}$ no state occurs twice (in other words, for $q_{i+1} = \delta(q_i, s[i+1])$, we have $q_{i+1} \neq q_i$). We include the null path $s = \varepsilon$ in this definition.

Fix an enumeration for $J : \mathcal{A}^* \to 1, 2, \ldots$, which induces an enumeration on the cover elements (shuffle ideals of $\mathcal{A}^*$). We shall construct a finitely supported separating hyperplane $\mathbf{w} \in \mathcal{X}$ corresponding to L. Abusing notation slightly, we will use w_i interchangeably with w_s, where $s = J^{-1}(i)$.

Let S be the set of all paths in A; observe that our definition of path together with Thm. 4.2 implies that S is finite.

Define

$$\eta(k) = \begin{cases} 1, & k = 0 \\ |\mathcal{A}| \left(\sum_{i=0}^{k-1} \eta(i) \right) + 1, & k > 0 \end{cases} \tag{10}$$

[4]Simon's original definition is different, but (as he showed) equivalent to this one.

and put

$$w_s = \begin{cases} +\eta(|s|), & s \in S \cap L \\ -\eta(|s|), & s \in S \setminus L \\ 0, & \text{else} \end{cases} . \tag{11}$$

It remains to show that this choice of $\mathbf{w}$ furnishes a linear separator for L. Consider some $u \in \mathcal{A}^*$ and let n be the length of the longest path $s \in S$ such that $u \in \mathrm{shuff}(s)$.

We make three key observations:

(1) u can belong to at most $|\mathcal{A}|^k$ cover elements $\mathrm{shuff}(s)$ with $|s| = n - k$,

(2) in particular, there is a unique $s(u) \in S$ s.t. $|s| = n$ and $u \in L$ iff $s(u) \in L$

(3)

$$|w_s| > \sum_{s' \in S : |s'| < |s|} |w_{s'}|$$

(the latter follows directly from the definition of η).

Thus the membership of u in L is determined by a single $s(u) \in S$; furthermore

$$\mathrm{sgn}(\langle \mathbf{w}, \boldsymbol{\varphi}(u) \rangle) = \mathrm{sgn}(w_{s(u)}) \tag{12}$$

and the latter is positive iff $u \in L$. □

On the other hand, Thm. 2.1 implies that any finitely supported hyperplane defines a piecewise testable language (it constructs the language as a boolean combination of the RLFC elements). Thus we obtain another characterization of the piecewise testable languages:

Theorem 4.4. *A language is piecewise testable iff it is linearly separable.*

Remark 4.5. We should clarify a somewhat subtle point. It is well known that not every function $f : \{0,1\}^n \to \{0,1\}$ is linearly separable when $\{0,1\}^n$ is embedded in $\mathbb{R}^n$ as the corners of the unit cube. For example, $f : \{0,1\}^2 \to \{0,1\}$ given by $f(x,y) = \mathrm{XOR}(x,y)$ is easily shown not linearly separable. Viewing a language as a boolean function on $\mathcal{X}$, the result we proved might seem to be at odds with the one just cited. Consider the language over $\mathcal{A} = \{\mathsf{a}, \mathsf{b}\}$ defined by

$$L = (\mathrm{shuff}(\mathsf{a}) \cup \mathrm{shuff}(\mathsf{b})) \cap \overline{(\mathrm{shuff}(\mathsf{a}) \cap \mathrm{shuff}(\mathsf{b}))}$$

(in other words, L is the XOR of the shuffle ideals of a and b). It is straightforward to verify that taking $w_i = [-1, 12, -4, -4, 2, 2]$ corresponding to the cover elements

$$[\emptyset, \mathrm{shuff}(\varepsilon), \mathrm{shuff}(\mathsf{ab}), \mathrm{shuff}(\mathsf{ba}), \mathrm{shuff}(\mathsf{a}), \mathrm{shuff}(\mathsf{b})]$$

(with $w_i = 0$ for all other coordinates) yields a linear separator for L (in fact, this was the maximum-margin separator obtained by applying SVM to a small random sample).

Notice that L was defined in terms of 2 cover elements/coordinates ($\mathcal{A}^*\mathsf{a}\mathcal{A}^*$ and $\mathcal{A}^*\mathsf{b}\mathcal{A}^*$) while the smallest subspace in which L is linearly separable is spanned 6-dimensional.

This is the key to resolving the apparent conflict: the infinite dimensionality of $\mathcal{X}$ allows the class of linearly separable languages to be closed under the boolean operations.

4.3 Efficient Kernel Computation

In order to complete the steps outlined in Sect. 3, we must produce an efficient computation of the kernel induced by the shuffle-ideal RLFC. For this RLFC, $K(s,t)$ counts the number of subsequences common to s and t, *not counting* multiple occurrences of the same subsequence:

$$K(s,t) = \sum_{u \in \mathcal{A}^*} [\![u \sqsubseteq s]\!] [\![u \sqsubseteq t]\!] \tag{13}$$

(where $[\![\cdot]\!]$ is used to assign truth values in $\{0,1\}$ to predicates contained within the bold brackets, and $x \sqsubseteq y$ means that x is a subsequence of y). Thus, $K(\mathsf{abc}, \mathsf{acbc}) = 8$ (the common subsequences being $\varepsilon, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{ab}, \mathsf{ac}, \mathsf{bc}, \mathsf{abc}$). A string with n distinct letters will have at least 2^n possible subsequences, so brute enumeration is not efficient.

The kernel defined by Lodhi et al. [12] is closely related to the shuffle-ideal kernel. Their kernel, $\tilde{K}$, has a parameter λ, and is defined as follows. Let $s, u \in \mathcal{A}^*$ with $u = s[i_1]s[i_2]\dots s[i_{|u|}]$ (that is, u is a subsequence of s). Denote this relationship by $u = s[\mathbf{i}]$, where $\mathbf{i} = [i_1, i_2, \dots, i_{|u|}]$, and define $\ell(\mathbf{i}) = i_{|u|} - i_1 + 1$. Then, for $s, t \in \mathcal{A}^*$, $\tilde{K}$ is defined to be

$$\tilde{K}(s,t) = \sum_{u \in \mathcal{A}^*} \sum_{\mathbf{i}: u = s[\mathbf{i}]} \sum_{\mathbf{j}: u = t[\mathbf{j}]} \lambda^{\ell(\mathbf{i}) + \ell(\mathbf{j})}. \tag{14}$$

Setting $\lambda = 1$ in $\tilde{K}$ almost yields the shuffle-ideal kernel, except the feature embedding counts multiple occurrences of a subsequence in a string, so $\tilde{K}(\mathsf{abc}, \mathsf{acbc}) = 9$. Lodhi et al. show how to compute their kernel efficiently. An efficient computation method for the shuffle-ideal kernel was communicated to us by Jonathan Derryberry [5]; it turns out to be similar in spirit to that of Lodhi et al.

For a letter $\sigma \in \mathcal{A}$ and a word $u \in \mathcal{A}^*$, define $\mathrm{last}_\sigma(u)$ to be 0 if σ does not occur in u and the largest i such that $u[i] = \sigma$ otherwise. For $s, t \in \mathcal{A}^*$, let

$$K'(s,t) = \sum_{u \in \mathcal{A}^+} [\![u \sqsubseteq s]\!] [\![u \sqsubseteq t]\!] \tag{15}$$

be the number of nonempty subsequences (without repetitions) common to s and t, and let

$$K_\sigma(s,t) = \sum_{u \in \mathcal{A}^+ \sigma} [\![u \sqsubseteq s]\!] [\![u \sqsubseteq t]\!] \tag{16}$$

be the number of such subsequences ending in σ. The following calculations are then almost immediate:

$$K'(s,t) = \sum_{\sigma \in \mathcal{A}} K_\sigma(s,t) \tag{17}$$

and

$$K_\sigma(s,t) = \begin{cases} 0, & \mathrm{last}_\sigma(s) = 0 \text{ or } \mathrm{last}_\sigma(t) = 0 \\ 1 + K'(s[1 : \mathrm{last}_\sigma(s) - 1], t[1 : \mathrm{last}_\sigma(t) - 1]), & \text{else} \end{cases}. \tag{18}$$

Then the shuffle-ideal kernel K, which always counts the empty string ε as a common subsequence, is given by $K(s,t) = K'(s,t) + 1$. A straightforward dynamic programming implementation of K achieves complexity $O(|s| \cdot |t|)$.

5 Conclusion

We have proposed a new paradigm for passively learning a regular language family $\mathcal{F}$ from labeled examples – namely, to embed the strings into a vector space via the mapping $\boldsymbol{\varphi}$ (which will depend on $\mathcal{F}$) and to apply kernel-SVM methods. Our approach has the advantage of entirely sidestepping the traditional techniques of building automata and merging states. This advantage, however, does not come for free: for each $\mathcal{F}$ that we would like to learn, we have to come up with the appropriate embedding $\boldsymbol{\varphi}$ (i.e., one that renders each member of $\mathcal{F}$ linearly separable), and to efficiently compute the kernel induced by $\boldsymbol{\varphi}$.

An obvious limitation of this (and any other) approach to PAC-learning regular languages is the inherent unpredictibility result, based on a cryptographic assumption [11]. In light of such limitations, the best one can hope for is to define a family of regular languages and exploit its structure to obtain learnability. Our methods provide a clean and unified theoretical framework for pursuing such results.

6 Future Work

Martin Zinkevich made the following observation: the family of sets $\mathcal{U} = \{\mathcal{A}^* a_1 \mathcal{A}^* a_2 \mathcal{A}^* \dots \mathcal{A}^* a_k \mathcal{A}^*\}$ (i.e., the shuffle ideals) has the property that any finite intersection of members of $\mathcal{U}$ is a finite union of members of $\mathcal{U}$:

$$\bigcap_{n=1}^{N} U_{i_n} = \bigcup_{m=1}^{M} U_{i_m}.$$

Call this the *finite conjunction-disjunction* property.

Thm. 4.3 says that all languages in the boolean algebra of $\mathcal{U}$ are linearly separable. Zinkevich and I conjecture that if $\mathcal{U}$ is a regular locally finite cover with the finite conjunction-disjunction property, then a language lies in the boolean algebra of $\mathcal{U}$ iff it is linearly separable under the embedding defined by $\mathcal{U}$. This generalizes thm. 4.3 to a much broader language family – the so-called dot-depth one family [6], which is the boolean algebra of sets of the form $\{\mathcal{A}^* u_1 \mathcal{A}^* u_2 \mathcal{A}^* \dots \mathcal{A}^* u_k \mathcal{A}^*\}$, where $u_i \in \mathcal{A}^+$. One interesting future direction is to prove the conjecture and to extend the kernel computation to the dot-depth one family. Another direction is to apply the methods proposed here to other language families known to be learnable (e.g., the k-reversible languages [3]) or perhaps even novel families. And of course, it would be good to obtain sample complexity bounds in terms of some measure of the complexity of the target language.

Acknowledgements

Many thanks to Yoram Singer for providing hosting and guidance at the Hebrew University in the summer of 2003. Martin Zinkevich corrected an error in the proof of Thm. 2.1; the author thanks him, Daniel Neill, and Avrim Blum for helpful discussions. Thanks to Jon Derryberry showing us an efficient algorithm for computing the kernel.

References

[1] D. Angluin. "On the complexity of minimum inference of regular sets." Information and Control. 39(3):337–350, 1978

[2] D. Angluin. “A note on the number of queries needed to identify regular languages.” Information and Control. 51(1):76–87, 1981

[3] D. Angluin. “Inference of reversible languages.” Journal of the ACM (JACM), 29(3):741–765, 1982

[4] Christopher J. C. Burges. “A Tutorial on Support Vector Machines for Pattern Recognition.” Data Mining and Knowledge Discovery. 2(2):121–167, 1998

[5] Jonathan Derryberry, private communication, 2004.

[6] Christian Glaßer and Heinz Schmitz. “The Boolean Structure of Dot-Depth One.” Journal of Automata, Languages and Combinatorics 6(4):437–452, 2001

[7] E. M. Gold. “Language identification in the limit.” Information and Control. 10(50):447–474, 1967

[8] E. M. Gold. “Complexity of automaton identification from given data.” Information and Control. 37(3):302–420, 1978

[9] N. Littlestone and M. Warmuth. “Relating data compression and learnability.” Technical report, University of California Santa Cruz, 1986

[10] Micheal Kearns, Ming Li, Leonard Pitt, and Les Valiant. “On the learnability of boolean formulae.” In 19th Annual Symposium on the Theory of Computing. ACM Press, 1987

[11] Micheal Kearns and Umesh Vazirani. An Introduction to Computational Learning Theory. The MIT Press, 1997.

[12] Huma Lodhi, Craig Saunders, Nello Cristianini, John Shawe-Taylor, Chris Watkins. “Text Classification using String Kernels.” NIPS 563–569, 2000

[13] M. Lothaire. Combinatorics on Words. Addison-Wesley, 1985.

[14] Rajesh Parekh and Vasant Honavar. “DFA Learning from simple examples.” Machine Learning 44:9–35, 2001

[15] L. Pitt and M. Warmuth. “The minimum consistent DFA problem cannot be approximated within any polynomial.” Journal of the Association for Computing Machinery, 40(1):95–142, 1993

[16] R. L. Rivest and R. E. Schapire. “Diversity-based inference of finite automata.” Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci., 78–87, 1987

[17] Dana Ron, Yoram Singer and Naftali Tishby. “On the learnability and usage of acyclic probabilistic finite automata.” Proceedings of the eighth annual conference on Computational learning theory, 31–40, 1995

[18] Imre Simon. “Piecewise testable events.” In H. Brakhage, editor, Automata Theory and Formal Languages, 2nd GI Conference, pages 214-222, Berlin, 1975. Springer-Verlag. Lecture Notes in Computer Science, 33

[19] A.N. Trahtman. “Piecewise and local threshold testability of DFA.” Lect. Notes in Comp. Sci., 2138:347–358, 2001

[20] B. A. Trakhtenbrot and Ya. A. Barzdin. Finite Automata: Behavior and Synthesis. North-Holland, 1973

[21] V. Vapnik. Statistical Learning Theory. Wiley-Interscience, New York, 1998

[22] Yechezkel Zalcstein. "Locally Testable Languages." Journal of Computer and System Sciences 6(2): 151-167, 1972