Mach Learn (2011) 82: 351–373
DOI 10.1007/s10994-010-5218-3

Languages as hyperplanes: grammatical inference with string kernels

Alexander Clark · Christophe Costa Florêncio · Chris Watkins

Received: 8 August 2008 / Revised: 10 July 2010 / Accepted: 31 August 2010 /
Published online: 25 September 2010

Abstract Using string kernels, languages can be represented as hyperplanes in a high dimensional feature space. We discuss the language-theoretic properties of this formalism with particular reference to the implicit feature maps defined by string kernels, considering the expressive power of the formalism, its closure properties and its relationship to other formalisms. We present a new family of grammatical inference algorithms based on this idea. We demonstrate that some mildly context-sensitive languages can be represented in this way and that it is possible to efficiently learn these using kernel PCA. We experimentally demonstrate the effectiveness of this approach on some standard examples of context-sensitive languages using small synthetic data sets.

Keywords Kernel methods · Grammatical inference

1 Introduction

A large percentage of data comes in the form of strings of symbols, sets of such strings make up a (formal) *language*. A natural machine learning problem is to infer a definition of a language from a set of positive examples of strings. The research field concerned with this type of problem is known as *grammatical inference*. This type of problem occurs in many areas. In data mining, for example, there may be a need to learn to recognise strings in particular flexible formats. Simple cases are email addresses or page formatting commands; more complex cases might be price-lists, postal addresses, or stock upgrades in free-text

Editor: Nicolò Cesa-Bianchi.

A. Clark (✉) · C. Watkins
Department of Computer Science, Royal Holloway, University of London, Egham TW20 0EX, UK
e-mail: alexc@cs.rhul.ac.uk

C. Costa Florêncio
Department of Computer Science, K.U. Leuven, Arenberg Campus III, Celestijnenlaan 200A, 3001 Heverlee (Leuven), Belgium
e-mail: Chris.CostaFlorencio@cs.kuleuven.be

broker reports. Many similar problems occur in bioinformatics and other branches of computational biology. A well-known example is finding rules for protein folding: the relation between a chain of amino acids that make up a polypeptide and the way it folds into a three-dimensional structure is still not completely understood.

The types of languages that we are interested in are in general infinite or very large, and thus the problem is one of finding a compact representation for the language. The most well-known and well-studied classes of representations are those in the Chomsky-Schützenberger hierarchy (Chomsky 1956): regular grammars, context-free grammars, context-sensitive grammars and unrestricted rewriting systems.

Grammatical inference algorithms for regular languages are now well understood, either using state merging algorithms for deterministic finite state automata (Clark and Thollard 2004), or using Hidden Markov Models (HMMs), the non-deterministic equivalent, with the EM algorithm. For context-free languages, some recent approaches have had some limited success (Starkie et al. 2004; Clark 2006, 2007; Clark and Eyraud 2007). However, it is known that the class of context-free grammars is extremely powerful and it is unlikely that algorithms exists that can learn the whole class of such languages. For context-sensitive languages, there are only a few positive theoretical results and no practical algorithms.

The particular problem domain that we are concerned with is that of natural language, and here the problem is much more acute. It is well-known that certain features of natural language cannot be described by context-free grammars but require the power of mildly context-sensitive grammars.

Rather than searching for algorithms to learn this whole class, which seems unlikely to be a successful research strategy, we follow another route: we consider an entirely different family of representations for which straightforward learning algorithms exist. The representation we use is a feature map from the set of strings into a Hilbert space. A language is then identified with the set of strings whose images lie in some region of this space. More specifically, in this paper we restrict ourselves to those languages defined by hyperplanes in the feature space.

We present and discuss computational experiments on a range of synthetic examples of languages chosen to be at different levels of the Chomsky hierarchy. We compare the performance of the new techniques with HMMs and PCFGs.

Our experiments may appear non-standard to machine learning researchers. We learn from just positive examples for two reasons: first, this is a standard approach in linguistically motivated theories of language learning, and second, because informative negative examples—"near misses"—may be difficult to generate, and they are rare in practice. We do of course generate negative examples as part of the test data.

A second way in which we depart from the conventions of kernel learning experiments is that we seek language descriptions as hyperplanes in the feature space, rather than as the more conventional half-spaces or clusters. Equality constraints are easy to interpret in simple cases, and can represent many of the particular languages we are interested in, though in other cases inequality constraints are necessary.

Thirdly, we use synthetic data generated from languages of known structure, rather than naturally occurring data. Our experiments are designed to explore the sorts of languages that this technique can learn rather than to demonstrate the utility of these methods on practical problems, an issue that we will address in future work. Still, our results are already highly relevant to some issues in language learnability (Gentner et al. 2006).

We seek learnable representations that are sufficiently expressive to represent these mildly context-sensitive languages. A key example which we shall return to is from Swiss German (Shieber 1985; Huybregts 1984), though similar phenomena occur in Dutch. Abstracting away from some details, Swiss German has some subordinate clauses where a

sequence of nounphrases can be followed by a sequence of verbs, of the same length, but where there are agreement constraints between the nouns and the verbs. Particular verbs require their corresponding nouns to be marked as a particular case, accusative or dative. Thus if we represent verbs by V_{acc}, V_{dat} where the subscript indicates the required case of the argument, and the nouns by N_{acc}, N_{dat}, the grammatical sentences are of the form $N_{acc}N_{dat}N_{dat}V_{acc}V_{dat}V_{dat}$, i.e. the sequence of nouns must agree with the sequence of verbs in the same order. No other orders are allowed, and there is no strict upper bound on the length of this construction, which is known as (serial) crossing dependencies. It is easy to see that this is not a context-free language through the application of a pumping lemma. It currently appears that all known non context-free phenomena in language[1] lie within the class of mildly context-sensitive languages, a class of languages defined by a number of weakly equivalent formalisms such as linear indexed grammars, tree adjoining grammars etc. These languages also include other non-context-free languages such as $\{a^n b^n c^n \mid n > 0\}$.

Modeling the acquisition of natural languages by children, or acquiring representations of natural language for NLP tasks will eventually require formalisms that can represent these structures, together with learning algorithms capable of acquiring them from observable data. It is also worth noting that some phenomena in computational biology, such as pseudoknots, also require expressive power beyond CFGs (Uemura et al. 1999).

Formally we situate our work in the context of classical grammatical inference from positive data: given an unknown language, and a finite sample of strings drawn from that language, and *without* any negative data, i.e., strings not in the language and marked as such, we wish to have an algorithm that can acquire a representation of the language that will enable us to determine whether a new string is in the language or not. The desiderata for such an algorithm include: reasonable observed sample complexity under natural distributions, polynomial computational complexity, insensitivity to small amounts of noise, and convergence over a sufficiently large class of languages.

1.1 Techniques

The familiar representations of languages are rewriting systems and automata of various types. These two families of representations converge at various points to yield the well-known Chomsky hierarchy. Unfortunately even low levels of the hierarchy are sufficiently powerful to represent cryptographically hard problems when considered as learning problems (Kearns and Valiant 1989).

A completely different approach is to represent languages through linear constraints on the substrings (Salomaa 2005). As a trivial example, consider the language over the alphabet $\{a, b\}$ consisting of equal numbers of as and bs in any order: example strings are $bbaa$, ab, $ababba$ etc. This is a context-free language, and can be defined either as a pushdown automaton or a more complicated context-free grammar.[2] However, we can also clearly represent this directly as the set of all strings that satisfy a certain linear equation on the occurrences of the symbols a and b, $L = \{u \in \{a, b\}^* \mid |u|_a = |u|_b\}$ where we write $|u|_a$ for the number of times a occurs in u.

Given this representation, a grammatical inference algorithm instantly suggests itself: map the strings into a certain vector space, and look for a low-dimensional subspace that

[1] We note a few exceptions whose status is questionable, such as suffix stacking in Old Georgian and a fraction of the Chinese number system.

[2] In fact, Asveld (2006) shows that the size of context-free grammars that generate just all permutations of n different symbols grows by a function exponential in n.

the data lie in. In this case the Parikh map (Parikh 1966) is sufficient, but other languages will require the use of counts of substrings of length greater than one. In this case we can use the implicit feature map defined by a string kernel, and, where appropriate, work in the dual representation. Our technique combines two well-understood techniques: kernel PCA (Schölkopf et al. 1998) together with string kernels (Watkins 2000; Lodhi et al. 2002).

1.2 Previous work

Complementary ideas are explored in Kontorovich et al. (2006), Cortes et al. (2007), however their approach is based on half-spaces. In contrast to our approach, the setting is supervised learning (using an SVM), so both positive and negative data is required. A generalization of the kernel used in Kontorovich et al. (2006), the regular finite cover embedding, has been shown to restrict the learnable class of languages to the regular languages. By contrast, our approach allows unsupervised learning of more expressive languages, more specifically of mildly context-sensitive languages.

1.3 Outline

The rest of the paper is arranged as follows. In Sect. 2 we will introduce some notation and define the various string kernels we will use. In Sect. 3 we will discuss the representational power of these hyperplanes, defined in terms of several different kernels, in language theoretic terms. In Sect. 4 we will discuss the formal learnability of the formalism in both a PAC learning framework and an identification in the limit framework and in Sect. 5 we will present some empirical work we have done, showing the effectiveness of our approach on several synthetic data sets.

2 Preliminaries

An alphabet Σ is a non-empty finite set of symbols, often called letters. The set of all strings over Σ, written Σ^*, is defined as the free monoid over Σ with null, the empty string, written as ϵ. The length of a string is $|w|$. A language L is a subset of Σ^*. We will write $\Sigma^k = \{w \in \Sigma^* \mid |w| = k\}$.

We will use the terminology and notation of Shawe-Taylor and Christianini (2004). Here $\mathbf{i}$ refers to a strictly ordered list of indices; i.e. $\mathbf{i}$ is a tuple $(i_1, i_2, \ldots, i_n)$ of positive integers, where $i_1 < i_2 < \cdots < i_n$. We will write $|\mathbf{i}| = n$ for the length of the tuple.

If u is of length n we can refer to the individual symbols as $u = u_1 \ldots u_n$. If $u, v \in \Sigma^*$, u is a subsequence of v if there are indices $\mathbf{i} = (i_1, \ldots i_{|\mathbf{i}|})$ with $1 \leq i_1 < \cdots < i_{|u|} \leq |v|$, such that $u_j = v_{i_j}$ for $j = 1 \ldots |u|$. Given a list of indices $\mathbf{i}$ we will define $v[\mathbf{i}]$ to be the subsequence $v_{i_1} v_{i_2} \ldots v_{i_{|\mathbf{i}|}}$ of v.

We write $u^R = u_n \ldots u_1$ for the reversal or mirror image of u.

2.1 Kernels

We will consider maps into Hilbert spaces of various types. Given a space X, in this case the set of strings Σ^*, and a mapping ϕ from $X \to H$, we define the kernel as the function from $X \times X \mapsto \mathbb{R}$:

$$K(u, v) = \langle \phi(u), \phi(v) \rangle. \tag{1}$$

Alternatively, the choice of kernel, which must be symmetric and positive definite, implicitly defines the mapping to the feature space. We used a number of different string kernels in our experiments, mainly using standard ones. Each kernel implicitly maps a string into a finite or infinite-dimensional feature space, typically indexed by substrings. The image of a string has coordinates that will correspond to a, possibly weighted, sum of the number of occurrences of that substring.

Fixed length subsequences kernel Features correspond to all non-contiguous subsequences of length k; where k is a hyperparameter of the kernel. The feature map is $\phi_u(w) = |\{\mathbf{i} : u = w[\mathbf{i}]\}|$, where $u \in \Sigma^k$.

Parikh kernel This is the special case of the fixed-length subsequences kernel with $k = 1$. The feature space thus has dimension $|\Sigma|$.

Gap weighted subsequences kernel All non-contiguous subsequences of length k where the gaps are weighted by the decay factor λ. $\phi_u(s) = \sum_{\mathbf{i}:u=s(\mathbf{i})} \lambda^{l(\mathbf{i})}$ where $|u| = k$. $l(i)$ is defined as $1 + i_{|\mathbf{i}|} - i_1$.

In the experiments reported here, we use $k = 2$ or $k = 1$.

3 Representations

We shall start by defining formally the class of planar languages, and examining the representational power of this formalism. For any given string kernel κ we can define the class of languages which are the preimages of finite-dimensional hyperplanes in the induced feature space. We call these the κ-planar languages, or ϕ-planar languages, where ϕ is the associated feature map.

Consider again the trivial language over the two letter alphabet $\Sigma = \{a, b\}$; $L_{ab} = \{u \in \Sigma^* \mid |u|_a = |u|_b\}$. This is an infinite language, which consists of all strings with equal numbers of as and bs. Example strings in the language are ab, ba, $aaaabbbabb$, $bbaa$, ... It is easy to show that this is not a regular language by applying a pumping lemma. Note that we have defined it explicitly as a linear relationship between two substring counts. If we consider the feature mapping defined by the Parikh kernel, which has exactly two dimensions, we can see that $\phi(L_{ab}) = \{(x, x) | x \geq 0\}$. Clearly these points lie in a hyperplane (a line in this case) in the feature space $\mathbb{R}^2$. Moreover, the preimage of the minimal hyperplane containing all the points of the language is exactly L_{ab}. More formally, we can define for any language L and feature map $\phi : \Sigma^* \to H$, where H is some Hilbert space, the hyperplane defined by (all affine combinations of) L as

$$H(L) = \left\{ \sum_i \alpha_i \phi(u_i) \in H \mid \exists u_i \in L\,,\ \alpha_i \in \mathbb{R},\ \text{such that } \sum_i \alpha_i = 1 \right\}$$

and the language $\hat{L}$ as the preimage of this hyperplane

$$\hat{L} = \{w \in \Sigma^* \mid \phi(w) \in H(L)\}.$$

A slight modification of this approach would be to consider all linear combinations: i.e. removing the constraint that the coefficients sum to one. This would make the dimension of the subspace 1 higher, and for some kernels would also change the representational power. For the Parikh kernel, all such languages would have to include the empty string.

Since we are interested in finite representations, we will say that for a finite set of strings $S = \{w_1, \ldots, w_n\}$:

$$L_\phi(S) = \left\{ w \in \Sigma^* | \exists \alpha_1 \ldots \alpha_n \in \mathbb{R} \sum_i \alpha_i = 1 : \phi(w) = \sum_i \alpha_i \phi(w_i) \right\}. \tag{2}$$

Note that $S \subseteq L_\phi(S)$. We can now say that a language L is ϕ-planar if and only if there is a finite set of strings S such that $L = L_\phi(S)$. Given a kernel κ we can define a planar language similarly. Note that for a given kernel κ the choice of feature map is of course arbitrary; however, for any two feature maps ϕ_1, ϕ_2 such that for all strings $u, v \in \Sigma^*$ we have $\langle \phi_1(u), \phi_1(v) \rangle = \langle \phi_2(u), \phi_2(v) \rangle$. We will also have that for all S, $L_{\phi_1}(S) = L_{\phi_2}(S)$, thus there is no ambiguity in using the terminology κ-planar languages.

For a given κ and a planar language L, we define the *rank* of the language to be the cardinality of the smallest set S such that $L_\phi(S) = L$.

Example 1 An important and obvious point to note is the dependence of the class on the kernel. Consider an arbitrary language L. Define a kernel κ_L as follows

$$\kappa_L(u, v) = 1 \quad \text{iff} \quad u, v \in L,$$
$$\kappa_L(u, v) = 0 \quad \text{otherwise.}$$

It is easy to verify that the language L is κ_L-planar. The implicit feature map here has one dimension and maps all strings in the language to the point (1) and all strings not in the language to the point (0). The hyperplane in this case is thus of dimension 0: a single point.

As is often the case with kernel methods, the selection of the kernel is of prime importance since it represents the prior knowledge about the problem. In this extreme case the question of learnability is of course trivial, but it generalizes to the case of more powerful kernels, which can be used to define larger and more interesting classes of languages.

Different kernels will enable different classes of languages to be defined. An important distinction is between kernels where the implicit feature map is injective, and those where it is not. The k-subsequence kernel is not injective, for any k. When $k = 1$, the two strings ab and ba are equivalent, when $k = 2$, the two strings $abba$ and $baab$ are equivalent, and such examples can be generated for any k. In practice, for sufficiently large values of k, the proportion of strings that are mapped to the same point in feature space is small (exponentially small in k).

The gap-weighted kernel weights features by polynomials in a parameter λ, corresponding to the numbers of gaps. Ignoring numerical issues, we can ensure that it is injective by setting the value of λ to be a suitable transcendental number, say $1/e$, which since it will not be the solution of any polynomial, means that the feature values will coincide only when the strings are identical.

We now discuss some elementary properties of the class of planar languages, focussing on language-theoretic rather than computational or learnability issues.

It is worth noting that the class of planar languages does not have nice closure properties. For example, the Parikh-planar languages are not closed under concatenation: both a^* and b^* are Parikh-planar, but $\{a^*b^*\}$ is not. Planar languages are closed under intersection, as we now demonstrate:

Proposition 1 (Clark et al. 2006b) *The intersection of two κ-planar languages is κ-planar.*

Proof Suppose that L_1 and L_2 are κ-planar for some kernel κ. Consider $H(L_1 \cap L_2)$, the least plane that contains the image of the intersection of L_1 and L_2.

Suppose $w \in L_1$ and $w \in L_2$. Clearly $\phi(w) \in H(L_1 \cap L_2)$. Conversely, suppose we have a w such that $\phi(w) \in H(L_1 \cap L_2)$. Now $H(L_1 \cap L_2) \subset H(L_1)$ so $\phi(w) \in H(L_1)$. Since L_1 is planar, $w \in L_1$. Similarly $w \in L_2$. So, $\phi(w) \in H(L_1 \cap L_2)$ if and only if $w \in L_1 \cap L_2$. Since L_1 and L_2 are planar, $H(L_1)$ has finite dimension, and thus so does $H(L_1 \cap L_2)$. Therefore $L_1 \cap L_2$ is planar. □

Obviously, the resulting hyperplane will be of lower dimensionality than the intersecting planes, unless these are identical.

Planar languages are generally not closed under reversal: one can construct a counterexample using a kernel that has features for initial substrings but not for terminal ones. For some kernels the corresponding class of planar languages *is* closed under reversal, the trivial case being the Parikh kernel, since it does not take the order of symbols into account. More interesting are kernels that are based on counts of all substrings, since for such kernels it is possible to find a bijection of the feature space that maps features ab to ba. Note that all the kernels considered in this paper fall into this category.

We now define the schema kernel, which has not been discussed before. We do not use it directly but just as a means to prove the injectivity of another kernel.

Definition 1 Let $\Sigma' = \Sigma \cup \{?\}$. A *schema* is any sequence $\sigma \in \Sigma'^{+}$. A *gapped* schema is an element of $\Sigma \times \{?\}^* \times \Sigma$, or an element of Σ.

Given a string u and a schema σ the *count* of the schema in a string is the number of times it matches, more formally $|\{i | \forall j = 1, \ldots |\sigma|, \sigma[i] =? \text{ or } \sigma[i] = u[i + j]\}|$.

Definition 2 (The **gapSchemas** kernel) The gapped schema feature map (**gapSchemas**) maps strings to vectors, where each feature is indexed by a gapped schema, and the value of the feature is the count of the schema in the string. The gapped schema kernel is the kernel based on the gapped schema feature map.

Example 3 For the **gapSchemas** kernel,

$$\phi(abba) = \begin{pmatrix} a:2 \\ b:2 \\ aa:0 \\ ab:1 \\ ba:1 \\ bb:1 \\ a?a:0 \\ a?b:1 \\ b?a:1 \\ b?b:0 \\ a??a:1 \end{pmatrix}.$$

Definition 4 The gap-weighted subsequences feature map (**gapWeighted**): All non-contiguous subsequences of length p where each occurrence is weighted exponentially by the number of gaps. This is adjusted by a hyperparameter λ. $\phi_u(s) = \sum_{\mathbf{i}:u=s(\mathbf{i})} \lambda^{l(\mathbf{i})}$ where $|u| = p$. $l(i)$ is defined as $1 + i_{|i|} - i_1$.

The gap-weighted subsequences feature kernel is the kernel based on the gap-weighted subsequences feature map.

Definition 5 The gap-weighted subsequences plus feature map (**gapWeighted**+): Combines **gapWeighted** and the Parikh kernel times λ.[3] We also define the associated kernel.

Example 6 For the **gapWeighted** kernel, with $p = 2$:

$$\phi(bab) = \begin{pmatrix} a: \lambda \\ b: 2\lambda \\ aa: 0 \\ ab: \lambda^2 \\ ba: \lambda^2 \\ bb: \lambda^3 \end{pmatrix}.$$

Proposition 7 *The* **gapSchemas** *kernel is injective.*

Proof By construction of the inverse function. Given a feature representation $h \in H$, and a $w \in \Sigma^*$ such that $\phi(w) = h$. Let l be the length of the longest schema with a non-zero value in h, which will be unique. Clearly $|w| = l$.

1. The first character of the string is the first character of the longest schema.
2. The nth character (for $n > 1$) contributes as first character to one of the schemas of length $l - n + 1$, but not to the schemas of length $l - n + 2$. So, let S_n be the multiset enumerating just all first characters of schemas of $l - n + 1$. Then the nth character is the only element of the set $S_n - S_{n-1}$. □

3.1 Injectivity of the **gapWeighted**+ kernel

The proof of the following proposition makes use of the existence of transcendental numbers. A transcendental number is any complex number that is not algebraic, that is, not the solution of a non-zero polynomial equation with integer (or, equivalently, rational) coefficients; some standard examples are π and e. The intuition behind the proof is that there is a correspondence between gapped schemas and the complex polynomials obtained from the **gapWeighted**+ kernel.

Proposition 8 *There exists an injective function f from* **gapSchemas** *feature representations to* **gapWeighted**+ *feature representations, for $p = 2$ and a transcendental value for λ.*[4]

Proof We write H_1 for the space of **gapSchemas** representations and ϕ_1 for the associated feature map, and H_2, ϕ_2 for the space of **gapWeighted**+ feature representations, and its map, respectively. We define f as follows: $f_{ab}(h) = h_{ab}\lambda^2 + h_{a?b}\lambda^3 + \cdots$, $f_a(h) = \lambda h_a$. By construction we can see that $\phi_2(w) = f(\phi_1(w))$. Suppose we have two elements h, h' of H_1, that lie in $\phi_1(\Sigma^*)$, such that $f(h) = f(h')$. This means that for every feature ab, we have

[3] We could use the standard Parikh kernel here, the λ factor is purely for technical reasons.

[4] A proof of injectivity for non-integral rational values for λ appeared in Clark and Watkins (2008).

$f_{ab}(h) = f_{ab}(h')$ and therefore $h_{ab}\lambda^2 + h_{a?b}\lambda^3 + \cdots = h'_{ab}\lambda^2 + h'_{a?b}\lambda^3 + \cdots$. This means that λ is the root of the polynomial $(h_{ab} - h'_{ab})\lambda^2 + (h_{a?b} - h'_{a?b})\lambda^3 + \cdots = 0$. Note that this is a polynomial since we will only have finitely many non-zero feature values for images of strings. Since λ is transcendental this means that all of the coefficients must be zero, i.e., $h_{ab} = h'_{ab}, h_{a?b} = h'_{a?b}, \ldots$. Therefore $h = h'$ and the map is injective. □

3.2 Expressive power

The relationship between the classes of planar languages and classes in the Chomsky hierarchy is minimal: for computable kernels, membership will be decidable. Even the class of planar languages defined by the most elementary kernel, the Parikh kernel, contains languages which are non-context free, while not including simple finite languages like $\{a, aa\}$.

In order to define notions of efficient learnability, which we will do in the next section, we need to be explicit as to the representation we use and the size of the representation. Each kernel defines a different representation class, and we consider the representation for a κ-planar language to be a set of strings of minimal size, where the size of a finite set of strings S is

$$\|S\| = \sum_{w \in S} |w|. \tag{3}$$

Though the minimal representation may not be unique, every minimal representation clearly has the same size. Our requirements for efficient learnability, defined in the next section, will then require sample complexity, or alternatively the size of the characteristic set, to be polynomial in the size of this minimal representation.

The Parikh kernel is clearly too limited to be of much interest. However kernels with richer feature spaces can represent interesting languages. For example, consider the classic example of a context-free language $L = \{a^n b^n | n \geq 0\}$. If we use the kernel κ_2, the subsequence kernel of length 2, then if $w \in L_{a^n b^n}$, clearly $|w|_{ba} = 0$. Thus using these features, this language can be represented through the equalities $|s|_a = |s|_b$ and $|s|_{ba} = 0$, without any recursive structure or center-embedding (Gentner et al. 2006). Since κ_2 has such features we will now be able to represent languages like $L_{a^n b^n}$ as hyperplanes in this richer feature space. The use of features corresponding to substrings of length greater than 2 increases the expressive power. For example, while κ_2 can express ordering constraints, κ_3 can express that a certain string must appear *between* two other strings, and so on. This increased expressivity comes at a price; the dimensionality of the feature space is $\mathcal{O}(|\Sigma|^k)$ for κ_k, and thus the amount of data required to learn a simple language can increase radically as k increases. However, one of the conclusions of our research is that $k = 2$ is surprisingly powerful. Intuitively, values of $k = 3$ might be necessary to model languages where substrings are constrained to lie between two other strings.

The relationship between k-testable languages and planar languages defined by the k-spectrum kernel is a useful illustration of the power of our technique.[5] The k-testable languages are those that are defined by a set of admissible k-length substrings. Clearly any k-testable language defined by n strings $u_1, \ldots u_n$, $|u_i| = k$, is also a planar language defined by the n-dimensional subspace spanned by these n substrings (we neglect here the problems of boundary symbols and prefixes and suffixes). But the class of planar languages contains not just the axis-aligned hyperplanes defined by each of these basis vectors, but also non-axis aligned hyperplanes.

[5]The class of k-testable languages has been shown to be learnable and has been applied in bioinformatics (Yokomori and Kobayashi 1998).

4 Learnability

In this section we discuss the learnability of the class of planar languages in a more formal way, by applying two different learnability criteria. First we will discuss efficient PAC-learnability (Valiant 1984). Secondly we will consider a classic criterion of grammatical inference, identification in the limit (Gold 1967), and one of its polynomial variant as defined by de la Higuera (1997).

We are interested in learning from finite positive data sets. A natural algorithm suggests itself: given a finite subset of L, say S, we can clearly define as our hypothesis $\hat{S}$, the preimage of all affine combinations of the sample points. We can construct the affine combination for the hyperplane by keeping a finite basis and adding to it when necessary.

Algorithm 1 presents this more formally.

Algorithm 1 SPAN learning algorithm

```
Inputs: kernel κ, training data S = {w_1, ... w_l}
U = {}
for i = 1 to i = l do
  if w_i ∉ L_κ(U) then
    U ← U ∪ {w_i}
  end if
end for
```

First, it is clear that if the samples are being drawn from a language L that is a κ-planar language, or a subset of a κ-planar language, then the hypothesis will converge to the least hyperplane containing the language, as by definition such hyperplanes are finite dimensional. Secondly, if the language L is a κ-planar language, then the hypothesized language will always be a subset of L, since the span of any subset of the language will always lie within the language. Thirdly, the hypothesized language will always include the observed points. Finally, if the language is planar, and thus of finite rank, say r, the total number of implicit errors of prediction that it makes is clearly at most r.

Note that in this algorithm we are not concerned with margin bounds, but rather with points that exactly lie in the hyperplane. This raises some issues about numerical precision. We can distinguish two cases; the first is where the kernel has integer or rational values. In this case we can perform all calculations exactly using arbitrary precision integer arithmetic and issues of numerical precision do not arise. Alternatively, we may have a kernel which produces real valued Gram matrices: in this case we will be working with finite precision floating point calculations, which may give rise to errors. As is customary in machine learning, we will not consider such problems, but rather focus on the more central and important issues of the generalisation properties of the algorithm given slightly idealistic assumptions about the accuracy of the computation.

4.1 PAC learnability

We will now look at the probabilistic learnability of this formalism. Note that we are learning hyperplanes from positive data, not the more normal situation where we learn half-spaces from positive and negative data. Nonetheless, the proofs are straightforward.

We recall the standard definitions from the PAC paradigm (Kearns and Vazirani 1994).

Definition 9 $\mathrm{err}_{f,\mathcal{D}}(h)$ denotes the probability that hypothesis h disagrees with the target function f on an instance x drawn from X_n according to $\mathcal{D}$. More formally, $\mathrm{err}_{f,\mathcal{D}}(h) = \Pr_{x\sim\mathcal{D}}[h(x) \neq f(x)]$.

Definition 10 A concept class F_n of size n is *PAC-learnable* if there is an algorithm A such that for every $f \in F$, any probability distribution $\mathcal{D}$, any error parameter $0 \leq \epsilon < 1$, and any confidence parameter $0 \leq \delta < 1$, A outputs hypothesis h such that $\Pr[\mathrm{err}_{f,\mathcal{D}}(h) \leq \epsilon] \geq 1-\delta$.

A concept class F is *efficiently* PAC-learnable if there exists a PAC-learning algorithm A for the class that takes a number of examples bounded by some polynomial in n, $1/\epsilon$ and $1/\delta$, and runs in time polynomially bounded by the size of the input data.

Note that since the input space consists of strings of variable length, we define the size of the data $w_1, \ldots w_m$ to be $\sum_I |w_i|$, and allow the learner to use time polynomial in this.

4.1.1 VC dimension

We assume familiarity with the notion of VC dimension (Vapnik and Chervonenkis 1971). It is a standard result that the VC dimension of hyperplanes of dimensionality r is $r+1$. However this refers to the classes of half-spaces defined in an r-dimensional space; that is to say, the hypothesis is the set of points that lie to one side of a hyperplane. In our case we are concerned with hypotheses that consist of the points that lie in a hyperplane, not necessarily through the origin, of dimension at most r defined by at most $r+1$ points.

Consider any set of $r+2$ points. Either these points lie in the same r-dimensional plane, in which case they cannot be shattered since one of these points p must be expressible as a linear combination of the $r+1$ other points, and therefore we cannot represent the hypothesis that includes those $r+1$ points but excludes p; or they do not, in which case no hypothesis can contain all of the points. Therefore the VC-dimension is at most $r+1$.

Since we can efficiently compute a consistent hypothesis and the VC dimension is equal to the number of strings in the basis, we have established the efficient PAC-learnability of the class from positive and negative data. Moreover, since we are interested in positive data only, and the result is distribution free, we can establish PAC-learnability with one-sided error as follows:

Theorem 11 *For a given fixed polynomially evaluable kernel κ, there is a polynomial p such that for all $\epsilon > 0$, all $\delta > 0$, for all κ-planar languages L of rank r, for all distributions D, such that $p_D(w) > 0$ implies $w \in L$, when algorithm A is provided with at least $p(r, 1/\epsilon, 1/\delta)$ examples drawn independent and identically distributed (IID) from D, A will output a hypothesis that is a subset of the target language and with probability at least $1-\delta$, $P_D(\hat{L} \setminus L) < \epsilon$.*

More precisely we can give an explicit polynomial bound. The minimal hyperplane algorithm is a mistake-bounded algorithm, as discussed above, and admits a sample-compression scheme in the sense considered in Floyd and Warmuth (1995), and from their Theorem 6 it follows that for any $0 < \epsilon, \delta < 1$ it is sufficient if we have a sample size (for any $0 < \beta < 1$) of

$$m \geq \frac{1}{1-\beta}\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + (r+1) + \frac{r+1}{\epsilon}\ln\frac{1}{\beta\epsilon}\right).$$

4.2 Identifiability in the limit

The notion of learnability we will apply here is known as identification in the limit (Gold 1967). Within this framework, a class of languages is considered learnable if there exists a (computable) function over sequences of input data that converges on a correct hypothesis after a finite amount of data (assuming all data is presented eventually).

4.2.1 Behavioural constraints on learners

Identification in the limit provides criteria for the success of a learning process, but grants total freedom to learners prior to convergence. Generally speaking it's desirable to be able to impose additional constraints, to guarantee 'rational' behaviour of the learner. Formally this simply means choosing a subset of possible learners.

Many different constraints have been studied in the literature, we define the ones relevant to the present discussion:

Definition 1 (Consistent learning) A learning function φ is *consistent* on $\mathcal{G}$ if for any $L \in \mathrm{L}(\mathcal{G})$ and for any finite sequence σ of elements of L, either $\varphi(\sigma)$ is undefined or $\sigma \subseteq \mathrm{L}(\varphi(\sigma))$.

Definition 2 ((Strong) Monotonicity) The learning function φ is *monotone increasing* or *strong monotonic* if for all finite sequences $\langle s_0, \ldots, s_n \rangle$ and $\langle s_0, \ldots, s_{n+m} \rangle$, whenever $\varphi(\langle s_0, \ldots, s_n \rangle)$ and $\varphi(\langle s_0, \ldots, s_{n+m} \rangle)$ are defined, $\mathrm{L}(\varphi(\langle s_0, \ldots, s_n \rangle)) \subseteq \mathrm{L}(\varphi(\langle s_0, \ldots, s_{n+m} \rangle))$.

Definition 3 (Incrementality) The learning function φ is *incremental* if there exists a computable function ψ such that $\varphi(\langle s_0, \ldots, s_{n+1} \rangle) \simeq \psi(\varphi(\langle s_0, \ldots, s_n \rangle), s_{n+1})$.

4.2.2 Efficient identification in the limit

Identification in the limit of a class guarantees the existence of learning algorithms for that class, but the learning problem is not necessarily tractable. Since we are interested in applications we need to specify further constraints on the learner. Ideally we would use some notion of polynomial identification in the limit. Several different definitions have been proposed (de la Higuera 1997; Yokomori 1991 among others), but at the present time no consensus exists as to which is best. The latter is the more restrictive one, so in order to obtain the strongest positive efficiency result for our approach, we choose to apply this definition.

Yokomori (1991) defines a class as *polynomial-time identifiable in the limit from positive data* if there exists a learning algorithm for that class such that both the number of explicit errors of prediction and the computation time it needs for any sequence of data are bounded by polynomials over the complexity of the representation (rank, in this case).

4.3 Identification in the limit of planar languages

We are now in the position to establish the following result:

Theorem 12 *For a polynomially evaluable kernel κ, the algorithm* SPAN *identifies in the limit the class of κ-planar languages in polynomial time and data.*

Proof For any plane of rank r there is a set C of strings such that $|C| = r$ and for any enumeration e of C, $|\text{SPAN}(e)| = r$ (trivial). It follows immediately that for any enumeration e of C, SPAN(e) defines a plane for L, and thus that C is a characteristic set for L.

Consider $C' = C \cup S$, where S is a finite subset of L. By definition of planar language and SPAN, S is in the language defined by the plane SPAN(e), so for any enumeration e' of C', SPAN(e') = SPAN(e), and both define a plane for L.

Therefore the learning function based on SPAN will, after encountering all elements from a characteristic set C, hypothesise L and will not diverge from this hypothesis. This proves identifiability in the limit of the class of planar languages.

The characteristic set has a size[6] polynomial (in fact linear) in the rank of the target plane, the learner need only change its mind as many times as the size of the characteristic set, and SPAN can be implemented to run in polynomial time.

A learner for planar languages based on SPAN is consistent and monotone increasing by definition: any datapoint not in the hyperplane is taken into account when SPAN generates a new hypothesis, so its hypotheses are always consistent with the data seen so far. It is monotone increasing since any hypothesis from SPAN at any given time is a hyperplane that includes the previously hypothesized hyperplanes. Incrementality is obvious: SPAN can generate a new hypothesis given just a new string and the previous hypothesis. This is true regardless of the way the hyperplane is represented, it need not be defined in terms of the set of basis strings. □

4.4 Finite elasticity

Learnability, and related properties like existence of a mind change bound, are largely determined by topological properties of the class under consideration. One such property is the existence of an *infinite ascending chain* of languages. This means that for some languages $L_0, L_1, \ldots, L_n, \ldots$ in that class, $L_0 \subset L_1 \subset \cdots \subset L_n \subset \cdots$. This implies the weaker property known as *infinite elasticity*:

Definition 13 ((In)finite elasticity (Wright 1989; Motoki et al. 1991)) A class $\mathcal{L}$ of languages is said to have *infinite elasticity* if there exists an infinite sequence $\langle s_n \rangle_{n \in \mathbb{N}}$ of sentences and an infinite sequence $\langle L_n \rangle_{n \in \mathbb{N}}$ of languages in $\mathcal{L}$ such that for all $n \in \mathbb{N}$, $s_n \notin L_n$, and $\{s_0, \ldots, s_n\} \subseteq L_{n+1}$.

A class $\mathcal{L}$ of languages is said to have *finite elasticity* if it does not have infinite elasticity.

Finite elasticity is a sufficient condition for learnability under two conditions, as shown in Wright (1989):

Theorem 14 (Wright) *Let $\mathcal{G}$ be a class of grammars for a class of recursive languages, where $G \in \mathcal{G}$ is at least semi-decidable. If* $\text{L}(\mathcal{G})$ *has finite elasticity, then $\mathcal{G}$ is identifiable in the limit.*[7]

[6] Here 'size' simply means cardinality of the set. It makes little sense to include the length of the strings in this definition, since the class includes for example the finite language with one element $a^{10^{100}}$.

[7] We slightly abuse notation here, by letting L denote both the language generated by a grammar, and a class of languages generated by a class of grammars.

Thus one route to proving learnability of a class is demonstrating it has finite elasticity, which has the added benefit of allowing one to easily define a conservative learning algorithm. In this context this is not necessary, however we do get nice closure properties for free.

The following theorem, from Kanazawa (1994, 1998), generalizes Theorem 14. Let Σ and Υ be two alphabets, a relation $R \subseteq \Sigma^* \times \Upsilon^*$ is said to be *finite-valued* just if for every $s \in \Sigma^*$, there are at most finitely many $u \in \Upsilon^*$ such that Rsu. If M is a language over Υ, define a language $R^{-1}[M]$ over Σ by $R^{-1}[M] = \{s \mid \exists u(Rsu \wedge u \in M)\}$.

Theorem 15 (Kanazawa) *Let $\mathcal{M}$ be a class of languages over Υ that has finite elasticity, and let $R \subseteq \Sigma^* \times \Upsilon^*$ be a finite-valued relation. Then $\mathcal{L} = \{R^{-1}[M] \mid M \in \mathcal{M}\}$ also has finite elasticity.*

In Kanazawa (1998) this property is used to prove finite elasticity of string languages by proving finite elasticity of their corresponding derivation (tree) languages, but it obviously has a wider range of applications.

Proposition 2 *If the feature space defined by κ has finite dimension, then the class of κ-planar languages has finite elasticity.*

Proof Suppose the class has infinite elasticity, with strings $s_1, \ldots$ and languages $L_1 \ldots$. If we define $S_n = H(\{s_0, \ldots, s_n\})$, then $S_{n-1} \subseteq L_n$, and $S_n \not\subseteq L_n$. Obviously, $S_0 \subset S_1 \ldots$, which constitutes an infinite ascending chain. Since S_n must be of greater rank than S_{n-1}, and every S_n is included in a language in the class, there can't be any bound on the rank of the planes for languages in the class. This is in contradiction with the hypothesis that the feature space has finite dimension. □

Not all planar languages are based on such a feature space, so planar languages with infinite elasticity exist. For example, any class of planar languages that contains all finite languages (c.f. the kernel based on all substrings) has an infinite ascending chain. It is easy to see that the fixed length subsequences-, **gapWeighted**- and **gapWeighted**+ planar languages meet the condition: they are based on the occurrence of substrings of bounded length, so the size of the alphabet imposes a bound on the number of distinct substrings, and thus on the dimensionality of the feature space.

It is straightforward to establish the following corollary (see Wright 1989):

Corollary 16 *Any finite union of classes of κ-planar languages where κ is finite dimensional has finite elasticity.*

Corollary 17 *Any class $\mathcal{L}$ such that for any language $L \in \mathcal{L}$, for some substitution σ and language $L' \in \mathcal{L}_\kappa$, where $\mathcal{L}_\kappa$ is a given class of planar languages, $L = \sigma[L']$, and whose corresponding hyperplanes have bounded dimensionality, has finite elasticity (and is thus learnable).*

Thus planar languages can be generalised to larger classes of learnable languages. Unfortunately, the naive learning algorithms for these classes will run in exponential time.

5 Experiments

In this section we will discuss some experimental work that we have done to verify the correctness of the algorithms and explore the practicalities of using this approach.

5.1 Implementation

We have described the algorithm above in terms of the primal representation in the feature space. In practice, it is more convenient to perform the computations in a dual representation using only kernel operations. For some of the kernels, the number of dimensions in the feature space is less than the number of data points; nonetheless, in the interest of clarity we work throughout with the kernel representation. The training phase of the algorithm follows a standard kernel PCA method (Shawe-Taylor and Christianini 2004).

1. Inputs: a kernel, a set of training data, a set of test data.
2. Compute Gram matrix of the training data.
3. Compute translated Gram matrix, with center at origin in feature space.
4. Compute k, the rank of the translated Gram matrix.
5. Compute the k eigenvectors and eigenvalues.
6. Compute the translated matrix of training-test products.
7. Project the test strings onto the hyperplane defined by the training data.
8. Compute perpendicular distance from test strings to hyperplane.
9. If this distance exceeds a threshold, label the data as negative, otherwise label it as positive.

This algorithm was implemented using MATLAB.[8] On all of the experiments reported here, the running times were only a few minutes. We found that the threshold was easy to set: generally the squared residuals were either very close to zero, (of the order of 10^{-10}), or greater than 0.1. A more principled way of setting these bounds would involve comparing the distance of strings in the training data to the plane to get an idea of the distribution of numerical errors, and to base the threshold on that. However, since we did not encounter these problems in any of our experiments under any settings,we did not pursue this avenue. However, it is worth noting that these problems might occur using for example, kernels where all of the norms are bounded.

5.2 Data

We generated some synthetic data sets to evaluate the potential of this approach. We selected a number of languages that have been proposed in the literature, generally from small alphabets. For comparison, we also evaluated it on one of the more complex grammars from the Omphalos context-free grammatical inference competition (Starkie et al. 2004); this is at the state of the art for context-free grammatical inference. The data is available from the competition website.[9]

For each of the languages we generated some positive data, by sampling from a natural distribution. For example, for the copy languages, we first generated a random length, and then created a random string by sampling from the uniform distribution over all strings of

[8] Source code for our implementation and data sets are available at http://www.cs.rhul.ac.uk/home/alexc/gisk/.

[9] www.irisa.fr/Omphalos/data-sets.html.

that length. We then duplicated it to create the sample string. All of the positive data was generated IID. The lengths of the strings were generally less than 20, with a few exceptions: the strings from the Omphalos data set are much longer than this.

For evaluation we need both positive and negative data. Generating negative data is far from trivial: simply generating random strings in a similar way does not produce a test set sufficiently difficult to distinguish the true hypothesis from a similar but incorrect one, without using astronomical amounts of data—the same problem was encountered by the organisers of the Omphalos competition (Starkie et al. 2004). We thus generated the negative data sets 50% from random strings from a uniform distribution over strings, and 50% drawn from languages that are close to the true one. Thus for example, when testing languages like **An** − **Dn**, we generated samples from $\{a^+b^+c^+d^+\}$ as well as from $\{a, b, c, d\}^+$.

For comparison, we also implemented two baseline systems, based on Hidden Markov Models, and Probabilistic Context Free Grammars. For the HMM system, we randomly initialised an HMM with a fully connected transition matrix and an explicit end-of-string transition for each state, for the PCFG system we used a CNF grammar. In both cases they were trained to convergence with respectively the Baum-Welch algorithm and the inside-outside algorithm. We then evaluated the test strings and labelled them as positive or negative according to whether the probability of the string was above a simple length-based threshold. Though slightly *ad hoc*, empirically we observed that this was sufficient to distinguish the language when the model structure was correct.

There are no practical baselines for learning context-sensitive languages.

5.3 Languages

We tried a number of well-studied languages from computational and mathematical linguistics, as well as some variations, see Table 1.[10] **Bracket** is the bracket (Dyck) language (i.e., as and bs are balanced), which is known to be context-free. The corresponding phenomenon in natural language is center embedding, which seems to exist only in a very restricted form. **Even** is the set of all strings from $\{a, b\}^*$ that are of even length, which is obviously a regular language. **ChinNr** is an abstract representation of Chinese number words (Radzinski 1991), **GermScramb** of German verb scrambling (according to Becker et al. 1992). **AnBmCnDm** is known to be mildly context-sensitive but not expressible by Linear Indexed Grammars (LIG). The same holds for **MultCop** with $k \geq 0$. **DepBranch**, the dependent branches language, is mentioned in Vijay-Shanker et al. (1987) as an example of a language that cannot be generated by LIG. Note that **An** − **En** is also known to be beyond Tree Adjoining Grammars (TAG), in contrast to **An** − **Dn** and **AnBnCn**. The languages used in our training data are actually variants, where a^n is replaced by $\{a, b\}^n$, b^n by $\{c, d\}^n$ and so on. This was done to ensure that there were an exponentially large number of distinct strings in the language of bounded length. Without this precaution, simple memorization algorithms could seem to perform well. **Mix** (Bach 1981) is another well-known example of a mildly context-sensitive language, and has been shown not to be expressible by TAG.

CrossDepDA is a copy language with just one copy w of v, where v and w have a disjunct alphabet. **CrossDepCS** is a copy language with just one copy, and a center symbol that marks the boundary between the two subwords, **CrossDepND** is a copy language with one copy and no center marker.

[10]The results of this experimental evaluation were first discussed in Clark et al. (2006a).

Table 1 Definitions of target languages used. The column labeled 'Class' states whether the language is regular (REG), context-free (CF), mildly context-sensitive (MCS) or context-sensitive (CS). We use the map z, defined as $z(a) = e, z(b) = f, z(c) = g, z(d) = h$. $\pi(u)$ allows any permutation of the string u. Languages with an asterisk have had additional hard negative examples generated

Name	Class	Definition	Example strings
Bracket	CF	$\{ab, avb, vw \mid v, w \in \textbf{Bracket}\}$	*aaabbb*, *ab*, *aabbab*
PalinDisj	CF	$\{vw \mid v \in \{a,b,c,d\}^*, w^R = z(v)\}$	*abcgfe*, *dh*, *bdhf*
Palin	CF	$\{vv^R \mid v \in \{a,b,c,d\}^*\}$	*aa*, *bdaadb*
Even	REG	$\{\{a,b,c\}^{2n} \mid n \in \mathbb{N}\}$	*cbbabc*, *acab*
ChinNr $*$	CS	$\{ab^{k_1} \ldots ab^{k_r} \mid k_1 > \cdots > k_r > 0\}$	*abbbbabbb*, *abbbabbab*
Mix	MCS	$\{s \in \{a,b,c\}^* : \lvert s\rvert_a = \lvert s\rvert_b = \lvert s\rvert_c\}$	*bac*, *babcac*
GermScramb	MCS	$\{\pi(w)v \mid w = z(v), v \in \{a,b,c,d\}^+\}$	*dbhf*, *bdacfghe*
AnBnCn $*$	MCS	$\{a^n b^n c^n \mid n > 0\}$	*bde*, *abdcfe*
An $-$ **Dn** $*$	MCS	$\{a^n b^n c^n d^n \mid n > 0\}$	*adfh*, *bbcdefhg*
An $-$ **En** $*$	CS	$\{a^n b^n c^n d^n e^n \mid n > 0\}$	*acegi*, *aadcfeghij*
DepBranch	CS	$\{a^n b^m c^m d^l e^l f^n \mid n = m + l \geq 1\}$	*aaaabcdddeeefffff*
MultCop	CS	$\{w^k \mid k > 0, w \in \{a,b\}^*\}$	*babbab*, *abaabaaba*
AnBmCnDm $*$	MCS	$\{a^n b^m c^n d^m \mid n, m > 0\}$	*acfh*, *abaabccdeffeehgh*
CrossDepDA	MCS	$\{vzf(v) \mid v \in \{a,b,c,d\}^*\}$	*dbhf*, *cbdcgfhg*
CrossDepCS	MCS	$\{wxw \mid w \in \{a,b,c,d\}^*\}$	*cxc*, *bdaxbda*
CrossDepND	MCS	$\{ww \mid w \in \{a,\ldots,h\}^*\}$	*cgcg*, *fcgefcge*
Omp4	CF	Omphalos problem 4	Omphalos website

5.4 Results

Table 2 displays the results of training the baseline models and kernel PCA with two different kernels on these languages.[11] We can see that in all but four cases the string kernel method performs very well, converging to a hypothesis with very small error, whereas the baseline methods overgeneralize. In particular for our motivating example from Swiss German, **CrossDepDA**, we see a zero error rate. The four cases in question are: **ChinNr** where the HMM model performs well by learning a simple regular approximation; **Even** where both of the baseline models correctly learn the hypothesis; **MultCop** which is a very hard language to learn (in fact one of the authors was unable to determine what language it was from the generated positive data alone) and **CrossDepND**, where in the absence of a center marker there are no features that can define the language. In these cases, where the string kernel method fails to produce an accurate hypothesis, it overgenerates significantly. In the case of **AnBmCnDm** the string kernel method overgeneralises slightly, but very plausibly by allowing empty strings (generalising > 0 to ≥ 0). The string kernel method when applied to **Bracket** learns merely the hypothesis that there are equal numbers of as and bs, but is incapable of learning that no prefix must violate the constraint that there are more bs than as. HMM does well on **Bracket**, contrary to expectations, but merely by modeling some local features, even though it is CF. This is due to the fact that it is difficult to generate test data

[11] Since the data sets are synthetic, it is not appropriate to compare the figures from different rows. The negative data has been generated to highlight the weaknesses of the various approaches.

Table 2 Test results on various data sets; training sets all of size 100. For each data set we report the percentage error rate separately for positive and negative data (lower is better). FP is the number of false positives as a percentage of the negative data, and FN is similarly the false negative rate. The kernels used are the 2-subsequence kernel and the 2-gap-weighted kernel with $\lambda = 0.5$. In both cases we added the Parikh kernel. The R column reports the dimension of the subspace, equivalently the rank of the data set in the feature space. We could not complete the PCFG experiments for the Omphalos data set because of the length of the strings

Language	$\lvert\Sigma\rvert$	+/− Test	PCFG		HMM		1 + 2-subseq			GapWeighted		
			FP	FN	FP	FN	FP	FN	R	FP	FN	R
Bracket	2	537/463	0	0	3.4	1.3	10.8	0	3	10.8	0	5
PalinDisj	8	505/495	0.8	0	4	8.1	0	0	20	0	0	30
Palin	4	510/490	6.1	0	83.5	2.9	16.1	0	14	16.1	0	14
Even	3	739/261	0	0	0	0	100	0	12	100	0	12
ChinNr	2	500/500	94.4	0	36.0	0	100	0	6	100	0	6
Mix	3	500/500	100	0	94.6	0	0	0	5	0	0	10
GermScramb	8	500/500	100	0	97.8	0.6	0	0	26	0	0	51
AnBnCn	6	509/491	8.8	0	20.4	0	0	0	17	0	0	25
An − Dn	8	501/499	6.4	0	46.5	0	0	0	24	0	0	38
An − En	10	500/500	38	0	37.5	0	0	0	32	0	0	54
DepBranch	6	500/500	6.4	0	6.4	0	0	0	5	0	0	14
MultCop	2	500/500	100	0	99.2	1.2	100	0	6	100	0	6
AnBmCnDm	8	507/493	50.4	0	49.5	0	0.8	0	31	0.8	0	42
CrossDepDA	8	505/495	4.2	0	5.7	4.3	0	0	20	0	0	36
CrossDepCS	5	515/485	3.3	2.1	8.7	2.1	0	0	20	8.5	0	27
CrossDepND	8	506/494	64.2	5.0	76.3	6.3	70.0	6.7	71	100	0	72
Omphalos	25	277/305	–	–	17.4	0.4	0	30	344	0	6	325

for **Bracket**. In order to penalize methods that depend on local features, bracket sequences should be used that represent heavily skewed trees.

The kernel approach performs well on **CrossDepDA** and **CrossDepCS**. Either the disjunct alphabet or the inclusion of a center marker are sufficient for the kernel method to perform well. Note that although both kernels score perfectly on **CrossDepDA**, the 1+2-subsequence kernel will give false positives with certain strings such as *abbafeef*: strings of this type are so rare that they don't show up in test sets of this size. The Omphalos data is from a much more complex grammar, and consists of much longer strings. We were therefore unable to use the PCFG algorithm, because the time complexity of the inside-outside algorithm is cubic in the length of the strings. Although neither of these kernels can induce an accurate representation, some structure has been learned, even though, as the high rank of the induced representations indicates, it overgenerated substantially.

In general, if the subspaces are of high rank with respect to the feature space, then this is a clue that the algorithm has failed to capture significant structure. Indeed **CrossDepND** illustrates this perfectly: the dimension of the feature space with an alphabet size of 8 is 72 for both kernels, and the rank is 71 or 72. We do not report results here for kernels with longer features; on the same data sets, with $k = 3$, we have a very large number of false negatives, because the rank of the languages becomes very much higher. For example, on the **GermScramb** data set, with $k = 3$ the false positive rate goes up to 94% with a rank of 94. With the longer features the rank of this language has increased to 914, so the span of the training data, which has size 100, is clearly insufficient.

6 Discussion

When the target language is a planar language for the kernel being used, the algorithm converges rapidly and exactly. Clearly, the dimension of the hyperplane (equivalently, the rank of the data in the feature space) is the key factor. Denoting this by r, it is clear that any exact representation of the hypothesis requires at least r points that are independent in the feature space. Empirically we observed that the hypothesis converged rapidly after the first r points. Conversely, when the language being learned is not exactly expressible as a hyperplane, the hypothesis converged to a superset of the target language. Thus in general, for sufficiently large amounts of data, we observe false positives but no false negatives. In some cases this superset was the whole monoid, Σ^*; for example the language **Even**. This is a good example of a comparatively simple, regular language that cannot be represented as a hyperplane by any of the kernels that we use here. Of course, it would be easy to rectify this by considering a kernel that also had features corresponding to $\phi_n(w) = 1$ iff $|w|$ is divisible by n. This language is easily learnable by the HMM baseline, surprising as it may seem. Overall, the string kernel method performs very well on these languages, and outperforms the baselines in general, especially in the context-sensitive cases.

The main computational bottleneck with this algorithm is the eigendecomposition of the *Gram* matrix, which means that the algorithm is cubic in the number of strings in the sample. However, there are more efficient algorithms which exploit the generally low rank of the *Gram* matrix in these applications (incomplete Cholesky factorisation) which allow algorithms that are linear in the amount of data. In our experiments we found that the learning was reasonably rapid on standard workstations for data sizes up to about 1000 strings, without any optimisation.

6.1 Related work

To the best of our knowledge, string kernels have not been used in this way before. The idea of using linear equations to define languages was discussed in Salomaa (2005), but the connection with string kernels has not been noted.

As discussed in the introduction, recent work by Kontorovich and others explored the use of string kernels for grammar induction; the approaches defined there work in a distribution-free paradigm from positive and negative data.

In Heinz (2010), the notion of *string extension learning* was introduced. The basic idea is that strings from a given language are mapped (extended) to an element of the generating grammar, so a learner can update its hypothesis by simply computing the union of its previous hypothesis and the yield of the string extension function applied to the current observation. Such string extension learners are obviously incremental, globally consistent, and locally conservative.

It is easy to see that any such function can be regarded as the embedding function of a string kernel, so the examples considered in Heinz (2010) can easily be dealt with in our approach. For example, the k factor languages are equivalent to the k-testable languages which can be learned with the k-spectrum kernel, the Strictly k-Piecewise languages are defined in terms of non-contiguous subsequences of length k which can be learned with the fixed length subsequences kernel. More formally, using the notation of Heinz (2010), given a string extension learner with a set A and a function $f : \Sigma^* \rightarrow \mathcal{P}_{fin}(A)$, a function from strings to finite subsets of A, we can define a vector space $H(A)$whose dimensions are indexed by the elements of A. We will let 1_a denote the unit vector in $H(A)$ corresponding to $a \in A$. We then replace the string extension map f by a map ϕ_f into $H(A)$ which is defined such that $\phi_f(w) \cdot 1_a = 1$ if $a \in f(w)$ and 0 if a is not in $f(w)$. For a given $G \subseteq A$, the language defined by G is defined as $L_f(G) = \{w \in \Sigma^* : f(w) \subseteq G\}$. It is easy to see that if $w \in L_f(G)$, then $\phi(w)$ lies in the $|G|$-dimensional hyperplane spanned by $\{1_a | a \in G\}$, and that any string mapping to a point in this hyperplane will also be in $L_f(G)$.

In terms of the results, there have been very few grammatical inference algorithms that have worked with representations capable of learning context-sensitive languages. The only such algorithm known to the authors was presented in Yoshinaka (2009) which generalized the notion of substitutable context-free language (Clark and Eyraud 2007) to include m-dimensional substitutable languages. This allows efficient induction of classes that contain mildly context-sensitive languages. However, a the author himself points out, even an extended version of his approach cannot deal with very simple mildly context-sensitive languages such as $\{a^n \# a^n | n \geq 1\}$, L_{reverse}, and L_{copy}.

In Becerra-Bonache and Yokomori (2004), an algorithm was presented that learns simple external contextual grammars (SECGs) from positive data, these generate mildly context-sensitive languages. This algorithm is exponential, but the later Oates et al. (2005) presents a polynomial-time learning algorithm for the same class. A careful reading of this work reveals that the setting is weaker than learning from positive data; a parameter d, the value of which is accessible to the learner, determines the maximum depth of derivations included in the characteristic set, and these are exhaustively generated. Thus, the learner is provided with some extra information not normally present in the identification in the limit paradigm. Finite elasticity of this class was shown in Becerra-Bonache (2006).

A fundamentally different approach to learning context-sensitive languages is discussed in Sempere (2008). There, the approach is restricted to learning from structured data, i.e., derivations as opposed to strings. Since the learner is supplied with more information than in our setting, the two approaches are not comparable.

Apart from these papers, some results are known that are purely theoretical and allow unbounded computation: Shinohara (1990) proves learnability for context-sensitive grammars with a length bound on the rules, but gives no algorithm. The only relevant results that we are familiar with is a body of work using neural networks (Chalup and Blair 1999). These papers show that under a suitable, carefully tuned training regime, various types of neural network are capable of learning some of the examples that we considered. However, these approaches do not generalise well, are hard to train, and have no theoretical guarantees.

The choice of kernel is clearly very important here: there are a number of other kernels that can be devised that might be able to learn other classes of languages. One of the surprising aspects of this approach is that even when the induced feature space is of quite small dimension, the representational power of the formalism is quite high.

Hyperplanes are in some sense the easiest sets of points to learn in a Hilbert space. While they are effective for some languages, there are other languages, such as $\{a^n b^m \mid n > m > 0\}$, which do not form hyperplanes but rather half-spaces. These of course can be learned using, for example, the generalised portrait algorithm. Similarly other structures such as manifolds, or clusters on hyperplanes, would be learnable using other techniques, and would define other classes of languages. Substantial amount of research has been carried out in learning such structures in other fields of machine learning. In this context, the work of Crammer and Singer (2003) is significant. They study the general problem of finding minimum volume ellipsoids that contain given points in a Bregmanian space. In the case of the models we consider here, it is important to consider examples that may be of arbitrary length, and thus, given the kernels we use, arbitrarily far away from the origin. Thus an enclosing volume, rather than a plane, leads to a very different set of solutions. Indeed, with these kernels it would lead to models that only define finite languages, which is undesirable for language theoretic considerations. Nonetheless this does suggest directions for future work: in particular it is unrealistic to assume that every example we receive will be grammatical. This would mean we should try to find a plane that does not necessarily include every single point we observe but rather allows a fraction to lie at some distance from the plane.

7 Conclusion

We have put forward a new representation for languages, as hyperplanes in an induced feature space, and shown that these languages can be efficiently learned from positive data. We have demonstrated that this class of languages includes linguistically interesting context-sensitive languages that are not learnable with current grammatical inference techniques.

Acknowledgements This work has been partially supported by the EU funded PASCAL Network of Excellence on Pattern Analysis, Statistical Modelling and Computational Learning.

References

Asveld, P. R. J. (2006). Generating all permutations by context-free grammars in Chomsky normal form. *Theoretical Computer Science*, *354*(1), 118–130.

Bach, E. (1981). Discontinuous constituents in generalized categorial grammars. In North east linguistics society (NELS 11) (pp. 1–12).

Becerra-Bonache, L. (2006). *On the learnability of mildly context-sensitive languages using positive data and correction queries*. Ph.D. thesis, Universitat Rovira i Virgili, Tarragona, Spain.

Becerra-Bonache, L., & Yokomori, T. (2004). Learning mild context-sensitiveness: Toward understanding children's language learning. In G. Paliouras & Y. Sakakibara (Eds.), *Lecture notes in computer science: Vol. 3264*. *ICGI* (pp. 53–64). Berlin: Springer.

Becker, T., Rambow, O., & Niv, M. (1992). *The derivational generative power of formal systems or scrambling is beyond LCFRS* (Tech. Rep. 92–38). Institute For Research in Cognitive Science, University of Pennsylvania.

Chalup, S., & Blair, A. D. (1999). Hill climbing in recurrent neural networks for learning the $a^n b^n c^n$ language. In *Proceedings of the sixth international conference on neural information processing* (pp. 508–513).

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, *2*, 113–124.

Clark, A. (2006). PAC-learning unambiguous NTS languages. In *Proceedings of the 8th international colloquium on grammatical inference (ICGI)* (pp. 59–71).

Clark, A. (2007). Learning deterministic context free grammars: the Omphalos competition. *Machine Learning*, *66*(1), 93–110.

Clark, A., & Eyraud, R. (2007). Polynomial identification in the limit of substitutable context-free languages. *Journal Machine Learning Research*, *8*, 1725–1745.

Clark, A., & Thollard, F. (2004). PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, *5*, 473–497.

Clark, A., & Watkins, C. (2008). Some alternatives to Parikh matrices using string kernels. *Fundamenta Informaticae*, *84*(3–4), 291–303.

Clark, A., Costa Florêncio, C., & Watkins, C. (2006a). Languages as hyperplanes: grammatical inference with string kernels. In *ECML, 17th European conference on machine learning* (pp. 90–101). Berlin: Springer.

Clark, A., Costa Florêncio, C., Watkins, C., & Serayet, M. (2006b). Planar languages and learnability. In *Proceedings of the international conference on grammatical inference* (pp. 148–160). Tokyo: Springer.

Cortes, C., Kontorovich, L., & Mohri, M. (2007). Learning languages with rational kernels. In *Lecture notes in computer science: Vol. 4539. Proceedings of the 20th annual conference on learning theory (COLT 2007)* (pp. 349–364). Heidelberg: Springer.

Crammer, K., & Singer, Y. (2003). Learning algorithms for enclosing points in Bregmanian spheres. In *16th annual conference on learning theory* (p. 388). Berlin: Springer.

de la Higuera, C. (1997). Characteristic sets for polynomial grammatical inference. *Machine Learning*, *27*(2), 125–138.

Floyd, S., & Warmuth, M. (1995). Sample compression, learnability, and the Vapnik-Chervonenkis dimension. *Machine Learning 21*(3), 269–304.

Gentner, T. Q., Fenn, K. M., Margoliash, D., & Nusbaum, H. C. (2006). Recursive syntactic pattern learning by songbirds. *Nature*, *440*, 1204–1207. DOI 10.1038/nature04675. http://www.isrl.uiuc.edu/~amag/langev/paper/gentner06songbirds.html.

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

Heinz, J. (2010). String extension learning. In *Proceedings of the 48th annual meeting of the Association for Computational Linguistics*. Uppsala, Sweden.

Huybregts, R. (1984). The weak inadequacy of context-free phrase structure grammars. In G. J. de Haan, M. Trommelen, & W. Zonneveld (Eds.), *Van Periferie naar Kern*. Dordrecht: Foris.

Kanazawa, M. (1994). *A note on language classes with finite elasticity* (Tech. Rep. CS-R9471). CWI, Amsterdam.

Kanazawa, M. (1998). *Learnable classes of categorial grammars*. *CSLI publications*, Stanford: Stanford University, distributed by Cambridge University Press.

Kearns, M., & Valiant, L. G. (1989). Cryptographic limitations on learning boolean formulae and finite automata. In *21st annual ACM symposium on theory of computation* (pp. 433–444). New York: ACM.

Kearns, M., & Vazirani, U. (1994). *An introduction to computational learning theory*. Cambridge: MIT Press.

Kontorovich, L., Cortes, C., & Mohri, M. (2006). Learning linearly separable languages. In *Algorithmic learning theory, 17th international conference* (pp. 288–303).

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, *2*, 419–444.

Motoki, T., Shinohara, T., & Wright, K. (1991). The correct definition of finite elasticity: Corrigendum to identification of unions. In *The fourth workshop on computational learning theory*. San Mateo: Morgan Kaufmann.

Oates, T., Amstrong, T., Becerra-Bonache, L., & Atamas, M. (2005). A polynomial time algorithm for inferring grammars for mildly context sensitive languages. In *Workshop on grammatical inference applications: successes and future challenges* (pp. 61–65). Edinburgh, Scotland.

Parikh, R. J. (1966). On context-free languages. *Journal of the ACM*, *13*(4), 570–581.

Radzinski, D. (1991). Chinese number-names, tree adjoining languages, and mild context-sensitivity. *Computational Linguistics 17*(3), 277–299.

Salomaa, A. (2005). *On languages defined by numerical parameters* (Tech. Rep. 663). Turku Centre for Computer Science.

Schölkopf, B., Smola, A., & Müller, K. R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, *10*(5).

Sempere, J. M. (2008). Learning context-sensitive languages from linear structural information. In A. Clark, F. Coste, & L. Miclet (Eds.), *LNAI: Vol. 5278. Proceedings of 9th international colloquium on grammatical inference ICGI'08* (pp. 175–186). Berlin: Springer.

Shawe-Taylor, J., & Christianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge: Cambridge University Press.

Shieber, S. M. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, *8*, 333–343.

Shinohara, T. (1990). Inductive inference of monotonic formal systems from positive data. In S. Arikawa, S. Goto, S. Ohsuga, & T. Yokomori (Eds.), *Algorithmic learning theory* (pp. 339–351). New York: Springer.

Starkie, B., Coste, F., & van Zaanen, M. (2004). The Omphalos context-free grammar learning competition. In *LNAI: Vol. 3264. International colloquium on grammatical inference*, Athens, Greece (pp. 16–27). Berlin: Springer.

Uemura, Y., Hasegawa, A., Kobayashi, S., & Yokomori, T. (1999). Tree adjoining grammars for RNA structure prediction. *Theoretical Computer Science*, *210*(2), 277–303.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, *27*(11), 1134–1142.

Vapnik, V. N., & Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, *16*(2), 264–280.

Vijay-Shanker, K., Weir, D. J., & Joshi, A. K. (1987). Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th annual meeting on Association for Computational Linguistics* (pp. 104–111). Morristown: Association for Computational Linguistics.

Watkins, C. (2000). Dynamic alignment kernels. In A. J. Smola, P. L. Bartlette, B. Schölkopf, & D. Schuurmans (Eds.), *Advances in large margin classifiers* (pp. 39–50). Cambridge: MIT Press.

Wright, K. (1989). Identification of unions of languages drawn from an identifiable class. In *The 1989 workshop on computational learning theory* (pp. 328–333). San Mateo: Morgan Kaufmann.

Yokomori, T. (1991). Polynomial-time learning of very simple grammars from positive data. In *Proceedings of the fourth annual workshop on computational learning theory*, University of California, Santa Cruz (pp. 213–227). New York: ACM Press.

Yokomori, T., & Kobayashi, S. (1998). Learning local languages and their application to DNA sequence analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(10), 1067–1079. DOI:http://dx.doi.org/10.1109/34.722617.

Yoshinaka, R. (2009). Learning mildly context-sensitive languages with multidimensional substitutability from positive data. In R. Gavaldà, G. Lugosi, T. Zeugmann, & S. Zilles (Eds.), *Lecture notes in computer science: Vol. 5809. ALT* (pp. 278–292). Berlin: Springer.