Languages as Hyperplanes: Grammatical Inference with String Kernels

Alexander Clark, Christophe Costa Florêncio, and Chris Watkins

Department of Computer Science, Royal Holloway, University of London, Egham
TW20 0EX

Abstract. Using string kernels, languages can be represented as hyperplanes in a high dimensional feature space. 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 it is possible to efficiently learn these using kernel PCA. We present some experiments demonstrating the effectiveness of this approach on some standard examples of context sensitive languages using small synthetic data sets.

1 Introduction

Much data consists of strings of symbols. A set of symbol strings is known as a *language*: a natural machine learning problem is to infer a definition of a language from a set of positive examples of strings in the language. This problem has been much studied as *grammatical inference*, and this is the term we will use. This type of problem occurs in many fields. 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 broker reports. Sequence annotation in bioinformatics is another application.

Grammatical inference algorithms for regular languages are now well understood, either using state merging algorithms for deterministic finite state automata, 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). However, it is well known that certain features of natural language cannot be described by context free grammars, and require the power of mildly context sensitive grammars, which cannot be learned with current techniques.

In this paper we consider a new approach to grammatical inference based on the identification of hyperplanes in high-dimensional feature spaces induced by string kernels. This is entirely different from techniques previously studied in computational linguistics. 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.

J. Fürnkranz, T. Scheffer, and M. Spiliopoulou (Eds.): ECML 2006, LNAI 4212, pp. 90–101, 2006.

Our experiments may appear non-standard to machine learning researchers. We learn from positive examples only, for two reasons: first, this is a standard approach in language-learning theory; and second, because informative negative examples — "near misses" — may be difficult to generate, and they are rare in practice in real data. We do 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. Already, these results are 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), 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 verbs by N_{acc}, N_{dat}, the grammatical sentences are of the form $N_{acc}N_{dat}N_{dat}V_{acc}V_{dat}V_{dat}$: 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. 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 currently observed 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 CF languages such as $\{a^n b^n c^n \mid n > 0\}$.

Modelling the acquisition of natural languages by children, or acquiring representations of natural language for NLP tasks will eventually require representations that can represent these structures, together with learning algorithms capable of acquiring them from observable data.

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

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

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, robustness to small amounts of noise in the strings, 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 give 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 & 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 surprisingly complicated context free grammar. But we can clearly directly represent this 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.

Looked at in 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 the data lie in. In this case the Parikh map (Parikh, 1966) is sufficient. Other languages will require the use of counts of substrings of length greater than one, and 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 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. A language L is a subset of Σ^*.

If u is of length n we can refer to the individual symbols as $u = u_1 \dots u_n$. If $u, v \in \Sigma^*$, u is a subsequence of v if there are indices $\mathbf{i} = (i_1, \dots i_{|\mathbf{i}|})$ with $1 \leq i_1 < \cdots < i_{|u|} \leq |v|$, such that $u_j = v_{i_j}$ for $j = 1 \dots |u|$. We write $u^R = u_n \dots u_1$ for the reversal or mirror image of u.

For $u, v \in \Sigma^*$ we will write $|u|_v$ for the number of times that v occurs in u as a non-contiguous substring. For example, if $\Sigma = \{a, b, c\}$, and $u = caab$, then $|u|_{ab} = 2, |u|_{cb} = 1$.

The choice of kernel defines the mapping to the feature space. We used a number of different kernels in our experiments. We will use the terminology and notation of (Shawe-Taylor & Christianini, 2004). Here $\mathbf{i}$ refers to a strictly ordered list of indices.

Fixed length subsequences kernel. All non contiguous subsequences of length k. The features are restricted to $|u| = k$.

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 λ. $\phi_u(s) = \sum_{\mathbf{i}:u=s(\mathbf{i})} \lambda^{l(\mathbf{i})}$ where $|u| = k$. $l(i)$ is defined as $1 + i_{|i|} - i_1$.

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

2 Representational Power

Before discussing the learnability of this class, we can look at the representational power of the formalism. For any given string kernel κ we can define the class of languages which are the pre-images of finite dimensional hyperplanes in the induced feature space. We call these the κ-planar languages. A language is κ-planar, if there exist strings $w_1, \dots, w_n$ such that

$$L = \{w \in \Sigma^* \mid \exists \alpha_1 \dots \alpha_n, \sum_i \alpha_i = 1 : \phi(w) = \sum_i \alpha_i \phi(w_i)\}$$

Different kernels will enable different 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. Other kernels however are normally injective. 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 will start with a 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 $ab, ba, aaaabbbabb, bbaa, \dots$. It is easy to show that this is not a regular language, by an application of a pumping lemma, or to define a push-down automaton or context free grammar that generates this. But the way we have written it is 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

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

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

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

In this case it is easy to see that $\hat{L} = L$.

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.

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. In this particularly trivial case we can see that for all $S \subset L$ with at least two distinct elements, $\hat{S} = L$. If we take a slightly less trivial language

$$L_{a^n b^n} = \{ a^n b^n \mid n > 0 \} \tag{1}$$

it is easy to see that if we use the same Parikh kernel, $\hat{L}_{a^n b^n} = L_{ab}$. Since the Parikh kernel is not injective and indeed any two permutations of the same string are mapped to the same point, the representational power of this is very limited since it cannot represent any order constraints.

Consider the kernel κ_2, the subsequence kernel of length 2. If $w \in L_{a^n b^n}$, then clearly $|w|_{ba} = 0$. Thus using these features this language can be represented as $|s|_a = |s|_b$ 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.

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. 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.

It is worth noting that the class of planar languages does not have nice closure properties. It is closed under reversal and intersection, but not in general under union, concatenation, or other standard operations (Clark et al., 2006).

3 Algorithm

We have described the algorithm informally 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 we work throughout with the kernel representation, for ease of use. The training phase of the algorithm follows a standard kernel PCA method (Shawe-Taylor & 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.

It is easy to establish that for a kernel that can be evaluated in polynomial time, and where the size of the representation is taken to be the rank of the plane, the class of all planar languages can be polynomially identified in the limit from positive data alone. Similarly, it can be proved that this class can be PAC-learned, with sample complexity polynomial in the rank of the language. A description of the theoretical aspects of this can be found in (Clark et al., 2006).

4 Experiments

This algorithm was implemented using MATLAB. 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, 10^{-10}, or greater than 0.1.

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.

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 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. Negative data is more of a problem. 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 a HMM with a fully connected transition matrix, and with an explicit end of string transition for each state, and 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.

4.1 Languages

We tried a number of well-studied languages from computational and mathematical linguistics, as well as some variations, see Table 1. **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

Table 1. Definitions of target languages used. The column labels 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	$\{\{a, b\}^{2n} \mid n \in \mathbb{N}\}$	*cbbabc*, *acab*
ChinNr $*$	CS	$\{ab^{k_1} \dots ab^{k_r} \mid k_1 > \cdots > k_r > 0\}$	*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	$\{a^n b^m c^m d^l e^l f^n \mid n = m + l \geq 1\}$	*aaaabcdddeeeffff*,
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	$\{ww \mid w \in \{a, \dots, h\}^*\}$	*cgcg*, *fcgefcge*
Omp4	CF	Omphalos problem 4	Omphalos website[2]

(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. If not, simple memorization algorithms could 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, and **CrossDepND** is a copy language with one copy, and no center marker.

4.2 Results

Table 2 displays the results of training the baseline models and kernel PCA with two different kernels on these languages.[3] 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

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

[3] Since the data sets are synthetic, it is not appropriate to compare the figures from different rows, since 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	$\|\Sigma\|$	+/- 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

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 midpoint symbol, 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 modelling some local features, even though it is CF.

The kernel approach performs well on **CrossDepDA** and **CrossDepCS**. Either the disjunct alphabet or the inclusion of a center symbol are sufficient for the kernel method to perform well. Note that though 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. As a result, we could not use the PCFG algorithm, because the time complexity of the inside outside algorithm is cubic in the length. Note that though neither of the 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.

5 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 languages.

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.

5.1 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.

In terms of the results, there have been very few grammatical inference algorithms that have worked with representations capable of learning context sensitive languages, ignoring purely theoretical results that allow unbounded computation. The only relevant results that we are familiar with is a body of work using neural networks (Chalup & Blair, 1999). These papers show that under a suitable, carefully tuned training regime, various types of neural network are capable of learning some of these examples. However, these approaches do not generalise well, and are hard to train.

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.

6 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.

Acknowledgments

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

Bibliography

Bach, E. (1981). Discontinuous constituents in generalized categorial grammars. *North East Linguistics Society (NELS 11)* (pp. 1–12).

Becker, T., Rambow, O., & Niv, M. (1992). *The Derivational Generative Power of Formal Systems or Scrambling is Beyond LCFRS* (Technical Report 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 $a^nb^nc^n$ language. *Proceedings of the Sixth International Conference on Neural Information Processing.* (pp. 508–513).

Clark, A., Costa Florêncio, C., Watkins, C., & Serayet, M. (2006). Planar languages and learnability. *International Colloquium on Grammatical Inference (ICGI).* Tokyo. to appear.

Gentner, T. Q., Fenn, K. M., Margoliash, D., & Nusbaum, H. C. (2006). Recursive syntactic pattern learning by songbirds. *Nature*, *440*, 1204–1207.

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

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

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

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

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

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

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

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

Starkie, B., Coste, F., & van Zaanen, M. (2004). The Omphalos context-free grammar learning competition. *International Colloquium on Grammatical Inference* (pp. 16–27). Athens, Greece.

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

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