Languages as Hyperplanes
Grammatical Inference with string kernels

Alexander Clark

Department of Computer Science
Royal Holloway, University of London

PASCAL Symposium, Bled
Acknowledgements

This is joint work with Chris Watkins, Christophe Costa Florêncio and Mariette Serayet.

PASCAL Pump-priming grant 2005-2006
Grammatical Inference with String Kernels
This is joint work with Chris Watkins, Christophe Costa Florêncio and Mariette Serayet.

PASCAL Pump-priming grant 2005-2006
Grammatical Inference with String Kernels
Executive summary

- One year project: budget 66K Euros
- Post-doc based at RHUL
- Other partner: Colin de la Higuera’s lab at St Etienne.

Results

- ECML 2006 paper that won “ECML Innovative contribution award”
- ICGI 2006 paper; one journal paper, another on the way.
- Code and data on website; results have been replicated elsewhere
- EPSRC grant submitted; rejected; will resubmit
One year project: budget 66K Euros
Post-doc based at RHUL
Other partner: Colin de la Higuera’s lab at St Etienne.
Results
- ECML 2006 paper that won “ECML Innovative contribution award”
- ICGI 2006 paper; one journal paper, another on the way.
- Code and data on website; results have been replicated elsewhere
- EPSRC grant submitted; rejected; will resubmit
- Invited talks at workshop at ICML 2007, CogSci 2007, Pascal internal events, ...
One year project: budget 66K Euros
Post-doc based at RHUL
Other partner: Colin de la Higuera’s lab at St Etienne.

Results
- ECML 2006 paper that won “ECML Innovative contribution award”
- ICGI 2006 paper; one journal paper, another on the way.
- Code and data on website; results have been replicated elsewhere
- EPSRC grant submitted; rejected; will resubmit
- Invited talks at workshop at ICML 2007, CogSci 2007, Pascal internal events, ...
One year project: budget 66K Euros
Post-doc based at RHUL
Other partner: Colin de la Higuera’s lab at St Etienne.
Results
- ECML 2006 paper that won “ECML Innovative contribution award”
- ICGI 2006 paper; one journal paper, another on the way.
- Code and data on website; results have been replicated elsewhere
- EPSRC grant submitted; rejected; will resubmit
- Invited talks at workshop at ICML 2007, CogSci 2007, Pascal internal events, …
Idea
Formal languages can be described as the pre-image of hyperplanes in a feature space defined by string kernels.

Question
If we use standard string kernels, are the classes of languages interesting?

Topics
- theoretical analysis
- practical experiments
Idea
Formal languages can be described as the pre-image of hyperplanes in a feature space defined by string kernels.

Question
If we use standard string kernels, are the classes of languages interesting?

Topics
- theoretical analysis
- practical experiments
Idea
Formal languages can be described as the pre-image of hyperplanes in a feature space defined by string kernels.

Question
If we use standard string kernels, are the classes of languages interesting?

Topics
- theoretical analysis
- practical experiments
The scientific question
Linguistics

Dividing question
Does First Language acquisition proceed primarily through domain-specific mechanisms/knowledge or via general-purpose mechanisms?

Useful to have learning models that have a clean separation of prior knowledge
The scientific question
Linguistics

Dividing question

Does First Language acquisition proceed primarily through domain-specific mechanisms/knowledge or via general-purpose mechanisms?

Useful to have learning models that have a clean separation of prior knowledge
Problem with classic language theory

**Palindrome language**

\[ L = \{ww^R \mid w \in \{a, b\}^*\} \]

**Copy language**

\[ L = \{ww \mid w \in \{a, b\}^*\} \]

Question: why is the copy language much more complex than the palindrome language, when pre-theoretically it is simpler?
Problem with classic language theory

Palindromes language

\[ L = \{ w w^R | w \in \{a, b\}^* \} \]

Copy language

\[ L = \{ w w | w \in \{a, b\}^* \} \]

Question: why is the copy language much more complex than the palindrome language, when pre-theoretically it is simpler?
Consider the well known Parikh map from strings to a vector of counts of each of the letters.
If $|\Sigma| = n$ then $\phi_P : \Sigma^* \rightarrow \mathbb{R}^n$.

Example: $\Sigma = \{a, b\}$

$\phi_P(aaabab) = (4, 2)$
$\phi_P(ab) = (1, 1)$

Parikh’s lemma (1966)
The image of a context free language under the Parikh map is semi-linear.
Consider the well known Parikh map from strings to a vector of counts of each of the letters. If $|\Sigma| = n$ then $\phi_P : \Sigma^* \rightarrow \mathbb{R}^n$.

**Example:** $\Sigma = \{a, b\}$

- $\phi_P(aaabab) = (4, 2)$
- $\phi_P(ab) = (1, 1)$

**Parikh’s lemma (1966)**

The image of a context free language under the Parikh map is semi-linear.
Let $\Sigma = \{a, b\}$

Consider $L = \{s \in \Sigma^* : |s|_a = |s|_b\}$ where $|s|_a$ is the number of $a$'s in $s$

$L$ consists of strings with equal numbers of $a$ and $b$

Examples $ab, ba, aabb, bababa, baab, \ldots$
Image of this language under the Parikh map

String in the language if and only if its image is on the line.
Planar Languages

**Definition**

For any feature map $\phi$ from $\Sigma^*$ to a Hilbert space $H$, for any finite subset $S = \{w_1, \ldots, w_n\} \subset \Sigma^*$. we define

$$L_\phi(S) = \{w \in \Sigma^* | \exists \alpha_i, \sum \alpha_i = 1 \land \sum_i \alpha_i \phi(w_i) = \phi(w)\}$$
Finite basis; finite rank of hyperplane.

\[ R = \{ w_1, \ldots, w_n \}, \| R \| = \sum_i |w_i| \]

Affine combination.

Rank of plane = \( |R| - 1 \), not necessarily through origin.

Learnable using elementary linear algebra.

Does a test point lie on the plane formed by the training points?

Assume exact model of computation and neglect numerical issues.

In practice we don’t find accuracy a problem (using standard techniques).

Use kernels
Finite basis; finite rank of hyperplane.
\[ R = \{ w_1, \ldots, w_n \}, \| R \| = \sum_i |w_i| \]

Affine combination.  
Rank of plane = \(|R| - 1\), not necessarily through origin.

Learnable using elementary linear algebra.  
Does a test point lie on the plane formed by the training points?

Assume exact model of computation and neglect numerical issues.  
In practice we don’t find accuracy a problem (using standard techniques).

Use kernels
Comments on formal definition

- Finite basis; finite rank of hyperplane. 
  \[ R = \{ w_1, \ldots, w_n \}, \| R \| = \sum_i |w_i| \]
- Affine combination.
  Rank of plane = \(| R | - 1\), not necessarily through origin.
- Learnable using elementary linear algebra.
  Does a test point lie on the plane formed by the training points?
- Assume exact model of computation and neglect numerical issues.
  In practice we don’t find accuracy a problem (using standard techniques).
- Use kernels
Finite basis; finite rank of hyperplane.
\[ R = \{ w_1, \ldots, w_n \}, \| R \| = \sum_{i} |w_i| \]

Affine combination.
Rank of plane = \(|R| - 1\), not necessarily through origin.

Learnable using elementary linear algebra.
Does a test point lie on the plane formed by the training points?

Assume exact model of computation and neglect numerical issues.
In practice we don’t find accuracy a problem (using standard techniques).

Use kernels
Theoretical analysis

- Brand new branch of formal language theory based on geometry
- Closure properties
- Injectivity: does $\phi(u) = \phi(v)$ imply $u = v$?
- Learnability; trivial to establish some identification in the limit results; also a PAC result
Given a polynomial kernel $\kappa$.

**Algorithm 1**

Training data $S = \{w_1, \ldots, w_n\}$. Given a new string $w$, compute the distance to the hyperplane spanned by $S$. If this is large (non-zero), then this is not in the language, if it is small (close to zero) then it is in the language.

**Theorem**

This algorithm PAC-learns the class of $\kappa$-planar languages with sample complexity $\frac{|R|}{\epsilon} \log \frac{|R|}{\delta}$.
Learnability 2
Simple IIL algorithm

Given a polynomial kernel \( \kappa \).

**Algorithm 1**

Training data an infinite presentation of the language \( S = \{ w_1, \ldots, w_n, \ldots \} \). Start with \( B = \{ \} \). At each step \( i \), if \( w_i \in L(B) \), do nothing. Otherwise \( B \leftarrow B \cup \{ w_i \} \).

**Theorem**

This algorithm polynomially identifies in the limit the class of \( \kappa \)-planar languages

- The number of errors at most \( |R| \);
- The representation is itself a characteristic set.
Formal properties
Every language is planar

Specific kernel
For any language $L$ define map
$$\phi_L(w) = 1 \text{ if } w \in L \text{ otherwise } \phi_L(w) = 0.$$  

Fact
$L$ is $\phi_L$-planar

- One dimensional feature space
- Kernel represents prior knowledge; which can be very detailed.
**Formal properties**

Every language is planar

**Specific kernel**

For any language $L$ define map 
\[ \phi_L(w) = 1 \text{ if } w \in L \text{ otherwise } \phi_L(w) = 0. \]

**Fact**

$L$ is $\phi_L$-planar

- One dimensional feature space
- Kernel represents prior knowledge; which can be very detailed.
A key point is whether the feature map is injective.

**Definition**

A kernel $\kappa$ is injective if the feature map is injective i.e. if $\phi(u) = \phi(v) \Rightarrow u = v$.

$p$-subsequence kernel is not injective for any $p$:

$\phi_1(ab) = \phi_1(ba)$, $\phi_2(abba) = \phi_2(baab)$, ...
A key point is whether the feature map is injective.

**Definition**

A kernel $\kappa$ is injective if the feature map is injective i.e. if $\phi(u) = \phi(v) \Rightarrow u = v$.

$p$-subsequence kernel is not injective for any $p$:

$\phi_1(ab) = \phi_1(ba), \phi_2(aba) = \phi_2(baab), \ldots$
The gap-weighted kernel is injective if

- $\lambda$ is transcendental.
- $\lambda = \frac{p}{q}$, $p$, $q$ coprime, $q > 1$.

In bioinformatics, $\lambda < 1$ so gaps are penalised.
Possibly $\lambda > 1$ might be better as it spreads out the data sets more.

Open question: what about $\lambda = 2$?
Theorem
The gap-weighted kernel is injective if
- $\lambda$ is transcendental.
- $\lambda = \frac{p}{q}$, $p, q$ coprime, $q > 1$.

- In bioinformatics, $\lambda < 1$ so gaps are penalised.
- Possibly $\lambda > 1$ might be better as it spreads out the data sets more.

Open question: what about $\lambda = 2$?
Not really experiments: demonstrations

- Generate some random positive training data from example language
- Generate some random test data;
  - Negative data is hard to generate
  - Uniform samples are too easy
  - Added ad hoc approximation to the real samples to make the test harder.
- Induce model
- Test on the test data
  - False Positive rate = false positives / number of negatives
  - False Negative rate = false negatives / number of positives
Languages

- Classic examples from language theory
- Various levels of Chomsky hierarchy
- Focussed particularly on natural languages
- Simple languages: short descriptions
Two baseline systems:

- Hidden Markov Model
  Non deterministic finite state automaton
- PCFG
  In CNF with every possible rule
- Trained to convergence with EM algorithm.
  Forward-backward algorithm/ inside outside algorithm
- Probability threshold for language membership
- There are no baselines for learning context sensitive languages.
### Experiments: Even and Brackets

GISK worse than baselines

<table>
<thead>
<tr>
<th></th>
<th>Even (Regular)</th>
<th>Even number of symbols</th>
<th>abcb, ba, babacc, aaaa</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Alphabet ( { a, b, c } )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>abcb, ba, babacc, aaaa</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Bracket (CF)</th>
<th>Balanced brackets</th>
<th>(), (), ()(()))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Alphabet ( { (, ) } )</td>
<td>()(()))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>PCFG</th>
<th>HMM</th>
<th>SUBS</th>
<th>GPWT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FP</td>
<td>FN</td>
<td>FP</td>
<td>FN</td>
</tr>
<tr>
<td>Even</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bracket</td>
<td>0</td>
<td>0</td>
<td>3.4</td>
<td>1.3</td>
</tr>
</tbody>
</table>
Experiments: Even and Brackets
GISK worse than baselines

<table>
<thead>
<tr>
<th>Even</th>
<th>Even number of symbols</th>
<th>abcb, ba, babacc, aaaa</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Regular)</td>
<td>Alphabet ${a, b, c}$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bracket</th>
<th>Balanced brackets</th>
<th>()(), (()()), (()(()))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(CF)</td>
<td>Alphabet ${(,)}$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>PCFG</th>
<th>HMM</th>
<th>SUBS</th>
<th>GPWT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FP</td>
<td>FN</td>
<td>FP</td>
<td>FN</td>
</tr>
<tr>
<td>Even</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bracket</td>
<td>0</td>
<td>0</td>
<td>3.4</td>
<td>1.3</td>
</tr>
</tbody>
</table>
Planar Languages not Learned by HMMs or PCFGs

\[ A = \{ a_1, \ldots, a_N \}, \ B = \{ b_1, \ldots \} \]

Equality languages

\[ L_3 = \{ A^n B^n C^n \mid n \geq 0 \} \]
\[ L_4 = \{ A^n B^n C^n D^n \mid n \geq 0 \} \]
\[ L_5 = \{ A^n B^n C^n D^n E^n \mid n \geq 0 \} \]

Results

<table>
<thead>
<tr>
<th></th>
<th>PCFG</th>
<th>HMM</th>
<th>1+2-subseq</th>
<th>GapWeighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L )</td>
<td>FP</td>
<td>FN</td>
<td>FP</td>
<td>FN</td>
</tr>
<tr>
<td>( L_3 )</td>
<td>8.8</td>
<td>0</td>
<td>20.4</td>
<td>0</td>
</tr>
<tr>
<td>( L_4 )</td>
<td>6.4</td>
<td>0</td>
<td>46.5</td>
<td>0</td>
</tr>
<tr>
<td>( L_5 )</td>
<td>38</td>
<td>0</td>
<td>37.5</td>
<td>0</td>
</tr>
</tbody>
</table>

Alexander Clark

GISK
Abstraction of Swiss German data (Shieber, 1985):

- Nouns with various cases $N_{acc}, N_{dat}$ . . .
- Verbs that require cases $V_{acc}, V_{dat}$ . . .
- Sentences consist of a sequence of nouns, followed by verbs, with cross serial dependencies.

$L = \{ N_{acc}N_{dat}N_{dat}V_{acc}V_{dat}V_{dat}, \ldots \}$
Copy languages
Three variants

Formal definition

\[ N = \{ N_1, \ldots, N_n \}, \quad V = \{ V_1, \ldots, V_n \}, \quad f : N \rightarrow V, \quad n = 4 \]

\[ L_{\text{copy}} = \{ wf(w) | w \in N^* \} \]

\[ L_{\text{copynd}} = \{ ww | w \in N^* \} \]

\[ L_{\text{copycs}} = \{ wxw | w \in N^* \} \]

Results

<table>
<thead>
<tr>
<th></th>
<th>PCFG</th>
<th>HMM</th>
<th>1+2-subseq</th>
<th>GapWeighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L )</td>
<td>FP</td>
<td>FN</td>
<td>FP</td>
<td>FN</td>
</tr>
<tr>
<td>( L_{\text{copy}} )</td>
<td>4.2</td>
<td>0</td>
<td>5.7</td>
<td>4.3</td>
</tr>
<tr>
<td>( L_{\text{copynd}} )</td>
<td>64.2</td>
<td>5.0</td>
<td>76.3</td>
<td>6.3</td>
</tr>
<tr>
<td>( L_{\text{copycs}} )</td>
<td>3.3</td>
<td>2.1</td>
<td>8.7</td>
<td>2.1</td>
</tr>
</tbody>
</table>
Copy languages
Three variants

Formal definition

\[ N = \{N_1, \ldots, N_n\}, \quad V = \{V_1, \ldots, V_n\}, \quad f : N \rightarrow V, \quad n = 4 \]

\[ L_{copy} = \{wf(w) \mid w \in N^*\} \]

\[ L_{copynd} = \{ww \mid w \in N^*\} \]

\[ L_{copycs} = \{wxw \mid w \in N^*\} \]

Results

<table>
<thead>
<tr>
<th></th>
<th>PCFG</th>
<th>HMM</th>
<th>1+2-subseq</th>
<th>GapWeighted</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FP</td>
<td>FN</td>
<td>FP</td>
<td>FN</td>
</tr>
<tr>
<td>(L_{copy})</td>
<td>4.2</td>
<td>0</td>
<td>5.7</td>
<td>4.3</td>
</tr>
<tr>
<td>(L_{copynd})</td>
<td>64.2</td>
<td>5.0</td>
<td>76.3</td>
<td>6.3</td>
</tr>
<tr>
<td>(L_{copycs})</td>
<td>3.3</td>
<td>2.1</td>
<td>8.7</td>
<td>2.1</td>
</tr>
<tr>
<td></td>
<td>8.5</td>
<td>0</td>
<td>0</td>
<td>27</td>
</tr>
</tbody>
</table>
Distributional kernels
Dealing with large alphabets

Assumption of a finite alphabet $\Sigma$ is too simplistic.

- Words have internal structure – sequence of phone(me)s, letters.
- Lexical structure – case, number, gender, conceptual structure
- Need some way of capturing this internal structure of the alphabet.
- This might be given a priori, or could be learned.
- Large alphabets are computationally intractable

Subkernel

Assume we have a kernel over $\Sigma$, $\kappa : \Sigma \times \Sigma \rightarrow \mathbb{R}$
Assumption of a finite alphabet $\Sigma$ is too simplistic.

- Words have internal structure – sequence of phone(me)s, letters.
- Lexical structure – case, number, gender, conceptual structure
- Need some way of capturing this internal structure of the alphabet.
- This might be given \textit{a priori}, or could be learned.
- Large alphabets are computationally intractable

\textbf{Subkernel}

Assume we have a kernel over $\Sigma$, $\kappa : \Sigma \times \Sigma \rightarrow \mathbb{R}$
Distributional kernels
Learning a kernel

Given two words *cat* and *dog* we can expect them to behave similarly based on their distribution. (Harris, Schuetze . . . )

- This can be learned by looking at the statistics of a large corpus.
- Normally, we derived distributional statistics (vectors), cluster them and then use the cluster labels.
- Now, we can use the distributional statistics directly.
- Modify the string kernel that to use similarity matrix between symbols dimensions represent combinations of dimensions in symbol feature space.
Given two words *cat* and *dog* we can expect them to behave similarly based on their distribution. (Harris, Schuetze . . .)

- This can be learned by looking at the statistics of a large corpus.
- Normally, we derived distributional statistics (vectors), cluster them and then use the cluster labels.
- Now, *we can use the distributional statistics directly.*
- Modify the string kernel that to use similarity matrix between symbols. Dimensions represent combinations of dimensions in symbol feature space.
Experiments with distributional kernel
Target language with large alphabet

- Target Language: \( L = \{ A^n B^n C^n | n \geq 0 \}, \)
  \( A = \{ a_1, \ldots, a_{30} \}, B = \{ b_1, \ldots, b_{30} \}, C = \{ c_1, \ldots, c_{30} \}, \)
- Large alphabet \( |\Sigma| = 90, \) so language has rank \( > 10^3. \)
- Training data of 500 samples
- Three test sets of size 1000
  - Uniform
  - Positive
  - Hard \( \{ A^* B^* C^* \} \)
- 2-subsequence kernel.
Experiments with distributional kernel

Basic approach

Target language is planar, but data is inadequate, so the generated language is a subclass of the target language

Results

<table>
<thead>
<tr>
<th>Test Set</th>
<th>Positive</th>
<th>Uniform</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN</td>
<td>890</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>TN</td>
<td>0</td>
<td>881</td>
<td>1000</td>
</tr>
<tr>
<td>TP</td>
<td>110</td>
<td>115</td>
<td>0</td>
</tr>
<tr>
<td>FP</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Need at least 20,000 data points to get good generalisation, but algorithms are cubic . . .
Target language is planar, but data is inadequate, so the generated language is a subclass of the target language.

<table>
<thead>
<tr>
<th>Test Set</th>
<th>Positive</th>
<th>Uniform</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN</td>
<td>890</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>TN</td>
<td>0</td>
<td>881</td>
<td>1000</td>
</tr>
<tr>
<td>TP</td>
<td>110</td>
<td>115</td>
<td>0</td>
</tr>
<tr>
<td>FP</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Need at least 20,000 data points to get good generalisation, but algorithms are cubic . . .
Simple distributional kernel

For a given symbol $a \in \Sigma$ consider the distribution of immediately adjacent symbols.

Add a distinguished boundary symbol

Gives a $2(|\Sigma| + 1)$-dimensional feature space.

Count frequencies:

$$\kappa(a, b) = \sum_{\sigma \in \Sigma} c(\sigma a)c(\sigma b) + c(a\sigma)c(b\sigma)$$

Normalise so $\kappa(a, a) = 1$.

Efficient algorithm linear in size of data, so we can use as much data as we want.
Learn a distributional kernel from 500 strings.
Learn a distributional kernel from 10,000 strings.
20-dimensional approximation
Because of noise in the Gram matrix, strings will not lie exactly in hyperplane.

Longer strings have larger norm: measure angle to hyperplane, rather than perpendicular distance.

Threshold is set from training data (better – held out data).

Perfect score: 0 FP, 0 FN.

Large amount of data with a $\mathcal{O}(n)$ algorithm and then a smaller amount of data with a $\mathcal{O}(n^3)$ algorithm.
Because of noise in the Gram matrix, strings will not lie exactly in hyperplane

Longer strings have larger norm: measure angle to hyperplane, rather than perpendicular distance

Threshold is set from training data (better – held out data)

Perfect score: 0 FP, 0 FN

Large amount of data with a $O(n)$ algorithm and then a smaller amount of data with a $O(n^3)$ algorithm
Other research in project

- Pre-image problem
- Transductions
- Learning with Noise
- Extracting relations
Strings have a monoid structure: $u \circ (v \circ w) = (u \circ v) \circ w$.

Trees are non associative $u(vw) \neq (uv)w$

General program: look for associative representations: linear transformations of vector spaces, syntactic monoid. (cf Lambek, 1958)

Traditional representations of languages aren’t learnable, but other may be.
Strings have a monoid structure: $u \circ (v \circ w) = (u \circ v) \circ w$.

Trees are non associative $u(vw) \neq (uv)w$.

General program: look for associative representations: linear transformations of vector spaces, syntactic monoid. (cf Lambek, 1958)

Traditional representations of languages aren’t learnable, but other may be.
We can define languages geometrically using hyperplanes in a feature space defined by string kernels.

These languages include classic examples of mildly context sensitive languages that occur in natural languages.

These can be efficiently learned from positive data alone.