Languages as Hyperplanes

Grammatical Inference with String Kernels

Alexander Clark

Department of Computer Science
Royal Holloway, University of London

Sussex NLP Seminar

Acknowledgements

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

We would like to acknowledge support from the EU Pascal Network of Excellence,
in the form of a 'pump-priming' grant 2005-2006 for
Grammatical Inference with String Kernels

Acknowledgements

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

We would like to acknowledge support from the EU Pascal Network of Excellence,

in the form of a 'pump-priming' grant 2005-2006 for *Grammatical Inference with String Kernels*

Acknowledgements

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

We would like to acknowledge support from the EU Pascal Network of Excellence,
in the form of a 'pump-priming' grant 2005-2006 for
Grammatical Inference with String Kernels

Outline

1. Motivation
   - First Language Acquisition
2. Planar Languages
   - Simple example
   - Formal definition
   - Learnability
3. Empirical results
   - Practical Issues
   - Results

Outline

1. Motivation
   - First Language Acquisition
2. Planar Languages
   - Simple example
   - Formal definition
   - Learnability
3. Empirical results
   - Practical Issues
   - Results

How do children learn language?

- Without explicit instruction
- Without correction (middle class Western families aside)
- Rapidly
  - after a small amount of data
  - after a small amount of time
- Some feedback on well-formedness of utterances
- All natural languages
  - Includes some languages that are not context free
  - Swiss German, Bambara

Caveat: no natural language in this talk!

How do children learn language?

- Without explicit instruction
- Without correction (middle class Western families aside)
- Rapidly
  - after a small amount of data
  - after a small amount of time
- Some feedback on well-formedness of utterances
- All natural languages
  - Includes some languages that are not context free
  - Swiss German, Bambara

Caveat: no natural language in this talk!

Two possible research strategies

- The high road
  - Choose a sufficiently powerful class: CFGs, TAGs, . . . , that includes the natural languages.
  - Try to find an algorithm for learning some of them
- The low road
  - Choose a formalism that is inherently learnable
  - Try to make it powerful enough to represent natural languages.

Two possible research strategies

- The high road
  - Choose a sufficiently powerful class: CFGs, TAGs, . . . , that includes the natural languages.
  - Try to find an algorithm for learning some of them
- The low road
  - Choose a formalism that is inherently learnable
  - Try to make it powerful enough to represent natural languages.

Grammatical inference

- Formal languages
- Positive data
- Unstructured examples
- No side information
- Polynomial bounds on data and computation
- Different assumptions about samples

Problem with language theory

Palindrome language

$L = \{ww^R | w \in \{a, b\}^*\}$

Copy language

$L = \{ww | w \in \{a, b\}^*\}$

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

Problem with language theory

Palindrome language

$$L = \{ww^R | w \in \{a, b\}^*\}$$

Copy language

$$L = \{ww | w \in \{a, b\}^*\}$$

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

Royal Holloway University of London

Learnable representations

- Deterministic Finite State Automata (Clark and Thollard, 2004)
- Semi-Thue Systems/ Substitutable languages (Clark and Eyraud, 2005)
- Planar Languages (Clark, Costa Florêncio and Watkins, 2006)

Outline

Simple example
Parikh map

Consider the well known Parikh map from strings to a vector of counts of each of the letters.
If $|\Sigma| = n$ then $\phi_P : \Sigma^* \to \mathbb{R}^n$.

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

$\phi_P(aaabab) = \binom{4}{2}$
$\phi_P(ab) = \binom{1}{1}$

Parikh's lemma

The image of a context free language under the Parikh map is semi-linear.

Simple example
Parikh map

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

$\phi_P(aaabab) = \binom{4}{2}$
$\phi_P(ab) = \binom{1}{1}$

Parikh's lemma

The image of a context free language under the Parikh map is semi-linear.

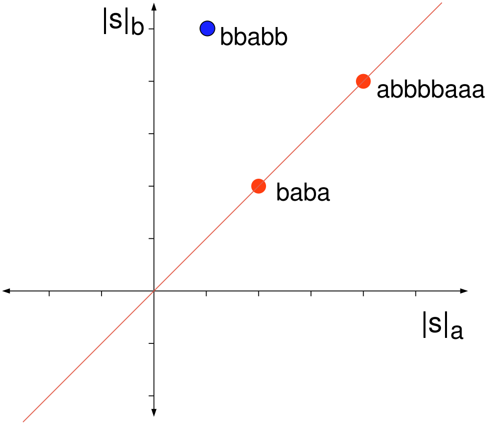
Simple Example

A context free language

- 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.

Outline

1. Motivation
   - First Language Acquisition
2. Planar Languages
   - Simple example
   - Formal definition
   - Learnability
3. Empirical results
   - Practical Issues
   - Results

Planar Languages

Definition

For any feature map ϕ from Σ^* to a Hilbert space H, for any finite subset $S = \{w_1, \dots, w_n\} \subset \Sigma^*$. we define $L_\phi(S) = \{w \in \Sigma^* | \exists \alpha_i, \sum \alpha_i = 1 \sum_i \alpha_i \phi(w_i) = \phi(w)\}$

Informally

Given a finite set of strings, a basis, we can define the language as the set of strings, whose images in feature space, that lie in the least hyperplane containing the images of the basis.

Royal Holloway University of London

Planar Languages

Definition

For any feature map ϕ from Σ^* to a Hilbert space H, for any finite subset $S = \{w_1, \ldots, w_n\} \subset \Sigma^*$. we define $L_\phi(S) = \{w \in \Sigma^* | \exists \alpha_i, \sum \alpha_i = 1 \sum_i \alpha_i \phi(w_i) = \phi(w)\}$

Informally

Given a finite set of strings, a basis, we can define the language as the set of strings, whose images in feature space, that lie in the least hyperplane containing the images of the basis.

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?

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?

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?

Kernels

We can use the kernel trick.

- We need to use feature spaces with large or infinite dimension.
- Explicitly computing ϕ may be intractable or impossible.
- It is enough to compute $\kappa(u, v) = \langle \phi(u), \phi(v) \rangle$. Basic linear algebra becomes slightly less basic linear algebra.

Outline

Learnability 1
Simple PAC algorithm

Given a polynomial kernel κ.

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 κ-planar languages.

Learnability 2

Simple IIL algorithm

Given a polynomial kernel κ.

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 κ-planar languages.

Formal properties

Every language is planar

Specific kernel

For any language L define map
$\phi_L(w) = 1$ if $w \in L$ otherwise $\phi_L(w) = 0$.

Fact

L is ϕ_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
$\phi_L(w) = 1$ if $w \in L$ otherwise $\phi_L(w) = 0$.

Fact

L is ϕ_L-planar

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

General purpose kernels

Each kernel defines an implicit feature space.

- Parikh kernel
- Spectrum kernel
- Subsequence kernel
- Gap-weighted kernel
- Discrete kernel: $\kappa_D(u, v) = \delta(u, v)$.
- All subsequences kernel

Small feature spaces tend to give overgeneralisation; huge feature spaces give poor or no generalisation.

General purpose kernels

Each kernel defines an implicit feature space.

- Parikh kernel
- Spectrum kernel
- Subsequence kernel
- Gap-weighted kernel
- Discrete kernel: $\kappa_D(u, v) = \delta(u, v)$.
- All subsequences kernel

Small feature spaces tend to give overgeneralisation; huge feature spaces give poor or no generalisation.

General program

e.g. Discrete kernel

- Defines a feature space
  Infinite dimensional feature space with one feature for each string $\phi(cat) = (0, 0, \ldots 0, 1, 0, \ldots)$
- Identify the planar languages with respect to this kernel
- All finite languages
  $L(S) = S$
- No generalisation: basis is the whole language

General program

e.g. Discrete kernel

- Defines a feature space
  Infinite dimensional feature space with one feature for each string $\phi(cat) = (0, 0, \ldots 0, 1, 0, \ldots)$
- Identify the planar languages with respect to this kernel
- All finite languages
  $L(S) = S$
- No generalisation: basis is the whole language

Royal Holloway University of London

General program

e.g. Discrete kernel

- Defines a feature space
  Infinite dimensional feature space with one feature for each string $\phi(cat) = (0, 0, \ldots 0, 1, 0, \ldots)$
- Identify the planar languages with respect to this kernel
- All finite languages
  $L(S) = S$
- No generalisation: basis is the whole language

General program

e.g. Discrete kernel

- Defines a feature space
  Infinite dimensional feature space with one feature for each string $\phi(cat) = (0, 0, \ldots 0, 1, 0, \ldots)$
- Identify the planar languages with respect to this kernel
- All finite languages
  $L(S) = S$
- No generalisation: basis is the whole language

General program

e.g. Discrete kernel

- Defines a feature space
  Infinite dimensional feature space with one feature for each string $\phi(cat) = (0, 0, \dots 0, 1, 0, \dots)$
- Identify the planar languages with respect to this kernel
- All finite languages
  $L(S) = S$
- No generalisation: basis is the whole language

p-subsequence kernel

kernel hyperparameters

p length of subsequences

$\Sigma = \{a, b\}$, $p = 2$

Features are scattered substrings of length p
(aa, ab, ba, bb)
$\phi(aaba) = (3, 2, 1, 0)$
$\phi(abba) = \phi(baab) = (1, 2, 2, 1)$

Parikh kernel is the 1-subsequence kernel.

p-subsequence kernel

kernel hyperparameters

p length of subsequences

$\Sigma = \{a, b\}$, $p = 2$

Features are scattered substrings of length p
(aa, ab, ba, bb)
$\phi(aaba) = (3, 2, 1, 0)$
$\phi(abba) = \phi(baab) = (1, 2, 2, 1)$

Parikh kernel is the 1-subsequence kernel.

Gap-weighted kernel

kernel hyperparameters

p length of subsequences
λ gap penalty

$\Sigma = \{a, b\}$, $p = 2, \lambda = 0.1$

Features are (aa, ab, ba, bb)
$\phi(aaba) = (1.11, 1.1, 1, 0)$

Modularity of kernels

We can combine kernels freely. or kernels κ_1, κ_2 with feature spaces H_1, H_2.

- $\kappa_1 + \kappa_2$ has feature space $H_1 \oplus H_2$
- $\kappa_1 \times \kappa_2$ has feature space $H_1 \otimes H_2$

Most of our work is with the kernel $\kappa_{GW+} = \kappa_2^G + \kappa_P$.

Injectivity

A key point is whether the feature map is injective.

Definition

A kernel κ 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:
$\phi_2(abba) = \phi_2(baab)$

Theorem

The gap-weighted kernel is injective if λ is transcendental.

Injectivity

A key point is whether the feature map is injective.

Definition

A kernel κ 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*:
$\phi_2(abba) = \phi_2(baab)$

Theorem

The gap-weighted kernel is injective if λ is transcendental.

Some planar languages

2-subsequence kernel

Examples

$\{a^n b^n | n \geq 0\}$

Not planar

$\{a^n b^n | n > 0\}$

$\{a^n b^m | n > m\}$

Some planar languages
2-subsequence kernel

Examples

$\{a^n b^n | n \geq 0\}$

Not planar

$\{a^n b^n | n > 0\}$
$\{a^n b^m | n > m\}$

Some planar languages

κ_{GW+}

Swiss german.

- Set of verb classes $V = \{v_1, v_2, \dots v_k\}$
- Set of noun classes $N = \{n_1, n_2, \dots n_k\}$
- $L = \{uf(u) | u \in V^*\}$

Examples: $v_1 v_2 v_3 n_1 n_2 n_3$
This is not a context free language.

L is planar for κ_{GW+}

$|u|_{v_i} = |u|_{n_i}$
$|u|_{v_i, v_j} = |u|_{n_i, n_j}$

Not planar for 2-subsequence kernel, because of congruent pairs.

Some planar languages

κ_{GW+}

Swiss german.

- Set of verb classes $V = \{v_1, v_2, \ldots v_k\}$
- Set of noun classes $N = \{n_1, n_2, \ldots n_k\}$
- $L = \{uf(u) | u \in V^*\}$

Examples: $v_1 v_2 v_3 n_1 n_2 n_3$
This is not a context free language.

L is planar for κ_{GW+}

$|u|_{v_i} = |u|_{n_i}$
$|u|_{v_i,v_j} = |u|_{n_i,n_j}$

Not planar for 2-subsequence kernel, because of congruent pairs.

Closure properties

Language theoretic properties of this class

Concatenation

$L_1 = \{a^n b^n | n \geq 0\}, L_2 = \{b^*\}$.
$L_1 L_2 = \{a^n b^m | m > n\}$ not planar.

Union

$L = \{a^n b^n\} \cup \{a^n b^{2n}\} \cup \{a^{2n} b^n\}$
generalises to $\{a^* b^*\}$

Intersection, reversal are the only closure properties.

Closure properties

Language theoretic properties of this class

Concatenation

$L_1 = \{a^n b^n | n \geq 0\}, L_2 = \{b^*\}$.
$L_1 L_2 = \{a^n b^m | m > n\}$ not planar.

Union

$L = \{a^n b^n\} \cup \{a^n b^{2n}\} \cup \{a^{2n} b^n\}$
generalises to $\{a^* b^*\}$

Intersection, reversal are the only closure properties.

Outline

1. Motivation
   - First Language Acquisition
2. Planar Languages
   - Simple example
   - Formal definition
   - Learnability
3. Empirical results
   - Practical Issues
   - Results

Implementation

Matlab sample code

Centering the Gram matrix

```
D = sum(K)/n; E = sum(D)/n; J = ones(n,1) * D;
K2 = K - J - J' + E * ones(n,n);
```

kernel PCA

```
k = rank(K2);
[V,L] = eigs(K2,k,'LM');
invL = diag(1./diag(L));
sqrtL = diag(sqrt(diag(L)));
invsqrtL = diag(1./diag(sqrtL));
Knew = K2' * V * invL * V' * K2;
```

Experimental setup

Not really experiments: demonstrations

- Generate some random positive training data from example language
- Generate some random test data;
  - Negative data is challenging
  - 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

Baselines

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

Outline

Results
Overview

- Some simple languages can't be learned by GISK method but can be by baselines
- Some languages are impossibly hard
- (Most important) Some interesting CF and CS languages can be learned by GISK.

Even (Regular)	Even number of symbols Alphabet $\{a, b, c\}$	abcb, ba, babacc, aaaa

Bracket (CF)	Balanced brackets Alphabet $\{(,)\}$	(), ()(), (()(()))

	PCFG		HMM		SUBS			GPWT		
	FP	FN	FP	FN	FP	FN	R	FP	FN	R
Even	0	0	0	0	100	0	12	100	0	12
Bracket	0	0	3.4	1.3	10.8	0	3	10.8	0	5

Experiments: Even and Brackets

GISK worse than baselines

Even (Regular)	Even number of symbols Alphabet $\{a, b, c\}$	abcb, ba, babacc, aaaa

Bracket (CF)	Balanced brackets Alphabet $\{(,)\}$	(), ()(), (()(()))

	PCFG		HMM		SUBS			GPWT		
	FP	FN	FP	FN	FP	FN	R	FP	FN	R
Even	0	0	0	0	100	0	12	100	0	12
Bracket	0	0	3.4	1.3	10.8	0	3	10.8	0	5

Planar Languages not Learned by HMMs or PCFGs

$$A = \{a_1, \ldots, a_N\}, B = \{b_1, \ldots\}, \ldots$$

Equality languages

$L_3 = \{A^n B^n C^n | n \geq 0\}$
$L_4 = \{A^n B^n C^n D^n | n \geq 0\}$
$L_5 = \{A^n B^n C^n D^n E^n | n \geq 0\}$

Results

	PCFG		HMM		1+2-subseq			GapWeighted		
L	FP	FN	FP	FN	FP	FN	R	FP	FN	R
L_3	8.8	0	20.4	0	0	0	17	0	0	25
L_4	6.4	0	46.5	0	0	0	24	0	0	38
L_5	38	0	37.5	0	0	0	32	0	0	54

Copy languages
Swiss german

Abstraction of Swiss German data (Shieber):

- Nouns with various cases $N_{acc}, N_{dat} \ldots$
- Verbs that require cases $V_{acc}, V_{dat} \ldots$
- 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

$$N = \{N_1, \ldots N_n\}, V = \{V_1 \ldots V_n\}, f : N \to V, \text{n= 4}$$
$$L_{copy} = \{wf(w) | w \in N^*\}$$
$$L_{copynd} = \{ww | w \in N^*\}$$
$$L_{copycs} = \{wxw | w \in N^*\}$$

Results

	PCFG		HMM		1+2-subseq			GapWeighted		
L	FP	FN	FP	FN	FP	FN	R	FP	FN	R
L_{copy}	4.2	0	5.7	4.3	0	0	20	0	0	36
L_{copynd}	64.2	5.0	76.3	6.3	70.0	6.7	71	100	0	72
L_{copycs}	3.3	2.1	8.7	2.1	0	0	20	8.5	0	27

Copy languages
Three variants

Formal definition

$N = \{N_1, \dots N_n\}, V = \{V_1 \dots V_n\}, f : N \to V$, n= 4

$L_{copy} = \{wf(w) | w \in N^*\}$

$L_{copynd} = \{ww | w \in N^*\}$

$L_{copycs} = \{wxw | w \in N^*\}$

Results

	PCFG		HMM		1+2-subseq			GapWeighted		
L	FP	FN	FP	FN	FP	FN	R	FP	FN	R
L_{copy}	4.2	0	5.7	4.3	0	0	20	0	0	36
L_{copynd}	64.2	5.0	76.3	6.3	70.0	6.7	71	100	0	72
L_{copycs}	3.3	2.1	8.7	2.1	0	0	20	8.5	0	27

Palindromes

Two variants

Languages

$$L_{palind} = \{wf(w^R) | w \in N^*\}$$
$$L_{palin} = \{ww^R | w \in N^*\}$$

Results

	PCFG		HMM		1+2-subseq			GapWeighted		
L	FP	FN	FP	FN	FP	FN	R	FP	FN	R
L_{palind}	0.8	0	4	8.1	0	0	20	0	0	30
L_{palin}	6.1	0	83.5	2.9	16.1	0	14	16.1	0	14

Palindromes

Two variants

Languages

$L_{palind} = \{wf(w^R)|w \in N^*\}$
$L_{palin} = \{ww^R|w \in N^*\}$

Results

	PCFG		HMM		1+2-subseq			GapWeighted		
L	FP	FN	FP	FN	FP	FN	R	FP	FN	R
L_{palind}	0.8	0	4	8.1	0	0	20	0	0	30
L_{palin}	6.1	0	83.5	2.9	16.1	0	14	16.1	0	14

Results

Very hard languages

two thousand thousand one thousand

Chinese numbers

$L = \{ab^{k_1} \dots ab^{k_r} | k_1 > \dots > k_r > 0\}$

Results

PCFG		HMM		1+2-subseq			GapWeighted		
FP	FN	FP	FN	FP	FN	R	FP	FN	R
100	0	99.2	1.2	100	0	6	100	0	6

Distributional kernels
Dealing with large alphabets

Assumption of a finite alphabet Σ 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 Σ, $\kappa : \Sigma \times \Sigma \to R$

Distributional kernels
Dealing with large alphabets

Assumption of a finite alphabet Σ 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 Σ, $\kappa : \Sigma \times \Sigma \to R$

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
- Kernel that uses similarity matrix between symbols dimensions represent combinations of dimensions in symbol feature space

Distributional kernels

Learning a kernel

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

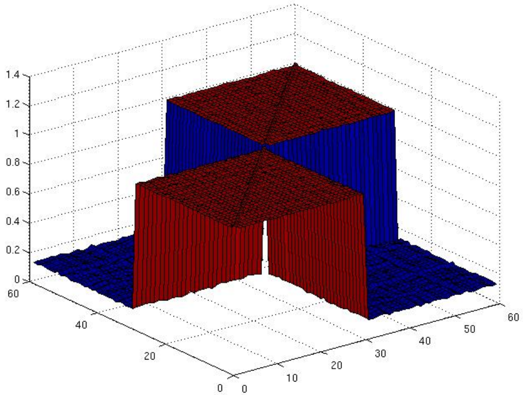
- This can be learned by looking at the statistics of a large corpus.
- Normally, we derived distributional statistics (vectors), cluster them and then use the cluster labels
- Now, we can use the distributional statistics directly
- Kernel that uses similarity matrix between symbols dimensions represent combinations of dimensions in symbol feature space

Experiments with distributional kernel

Preliminary only

- Target Language: $L_{copy} = \{wf(w) | w \in N^*\}$, $|N| = 30$
  1000 samples
- Four test sets of size 1000
  - Uniform
  - Positive
  - Hard $\{N^k V^k\}$
  - Very hard $\{w\pi(f(w))\}$
- Distributional kernel trained on extra 10,000 strings
- Approximate hyperplane by all eigenvalues above a threshold.

Learned Gram matrix
distributional kernel



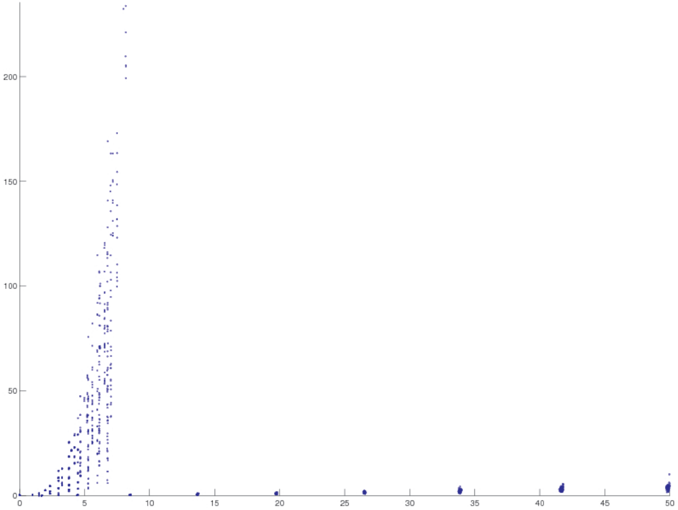
Experiments with distributional kernel
Results before tuning

Rank 833 for regular and 377 for dist kernel.

Results

TestSet	Distributional		Regular	
	FP	FN	FP	FN
Positive	0	343	0	803
Uniform	0	0	0	0
Hard	0	0	0	0
Very Hard	0	25	165	0

Scatter Plot



Previous work

- Kontorovitch: learning linearly separable languages.
  - Learning from positive and negative examples
  - Locally testable languages (subclass of regular languages)
- Salomaa: defining languages by numerical equations. Purely theoretical; no learning, no discussion of language theoretic power, no kernels.

Royal Holloway University of London

Previous work

- Kontorovitch: learning linearly separable languages.
  - Learning from positive and negative examples
  - Locally testable languages (subclass of regular languages)
- Salomaa: defining languages by numerical equations. Purely theoretical; no learning, no discussion of language theoretic power, no kernels.

Interfaces

Pure speculation

Linear representations of

- Semantics: LSA from bag of words
- Sound: Fourier kernels

Natural interface between a linear representation of syntax and linear models of the inputs and outputs.

Critical review
What are the weaknesses?

- Polynomial algorithms but cubic in number of sentences.
- Poor closure properties
- No experiments on real data (yet).
- Not a magic bullet; might need to be combined with another learning method.
- Useless – doesn't produce any structure.

Summary

- We can define languages geometrically using hyperplanes in a feature space.
- These languages include classic examples of mildly context sensitive languages that occur in natural languages.
- These can be efficiently learned from positive data alone.

- Future work
  - Learning with manifolds, hyper-ellipsoids
  - Learning with noise

Summary

- We can define languages geometrically using hyperplanes in a feature space.
- These languages include classic examples of mildly context sensitive languages that occur in natural languages.
- These can be efficiently learned from positive data alone.

- Future work
  - Learning with manifolds, hyper-ellipsoids
  - Learning with noise