Congruence based approaches

Learnable representations for languages

Alexander Clark

Department of Computer Science
Royal Holloway, University of London

August 2010
ESSLLI, 2010

Outline

Distributional learning

Congruence classes
- Syntactic monoid
- Canonical CFG

Algorithms
- Substitutable languages
- MAT learner
- NTS languages

Conclusion

Context free grammar

Tuple $\langle \Sigma, V, P, S \rangle$

- V is a set of non-terminals
- $S \in V$ is a start symbol
- P is a set of productions of the form $V \times (V \cup \Sigma)^*$, which we write as $N \to \alpha$

Context free grammar

Tuple $\langle \Sigma, V, P, S \rangle$

- V is a set of non-terminals
- $S \in V$ is a start symbol
- P is a set of productions of the form $V \times (V \cup \Sigma)^*$, which we write as $N \rightarrow \alpha$

- Derivation relation: $\beta N \gamma \rightarrow \beta \alpha \gamma$ when $N \rightarrow \alpha \in P$
- $L(G, N)\{w | N \stackrel{*}{\Rightarrow} w\}$
- $L(G) = \{w | S \stackrel{*}{\Rightarrow} w\}$

Almost Chomsky normal form

All rule are of the form

- $N \rightarrow PQ$
- $N \rightarrow a$
- $N \rightarrow \lambda$

- Multiple start symbols: $G = \langle \Sigma, V, P, I \rangle$ $I \subseteq V$ is a set of initial symbols.
- $L(G) = \bigcup_{S \in I} L(G, S)$

Distributional learning

Several reasons to take distributional learning seriously:

- Cognitively plausible (Saffran et al, 1996, Mintz, 2002)
- It works in practice: large scale lexical induction (Curran, J. 2003)
- Linguists use it as a constituent structure test (Carnie, A, 2008)
- Historically, PSGs were intended to be the output from distributional learning algorithms:

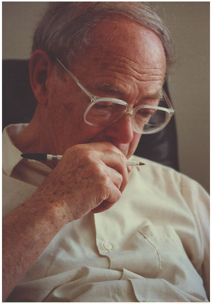
Chomsky (1968/2006)

“The concept of "phrase structure grammar" was explicitly designed to express the richest system that could reasonable be expected to result from the application of Harris-type procedures to a corpus.”

Distributional Learning

Zellig Harris (1949, 1951)

Here as throughout these procedures X and Y are substitutable if for every utterance which includes X we can find (or gain native acceptance for) an utterance which is identical except for having Y in the place of X



Empirical work on Distributional Learning

Real corpora

- Sample is not just of grammatical sentences
- Also semantically well-formed
- Also “true” in some non-technical sense
- Empirical distribution is very complex

Distributional similarity in real corpora often reflects semantic relatedness.

Various notions of context

Local syntactic context

Immediately preceding and following word
"If the candidate has an outstanding examination result"
Word "has" – context (candidate, an)

Wide bag-of-words context

Skip stop words
Set of words occurring in the same sentence/discourse.

Probabilistic models

Vector of counts of frequent words
Schütze

Distribution

Full context

Context (or *environment*)

A context is just a pair of strings $(l, r) \in \Sigma^* \times \Sigma^*$.
$(l, r) \odot u = lur$
$f = (l, r)$.
Special context (λ, λ)

Given a language $L \subseteq \Sigma^*$.

Distribution of a string

$C_L(u) = \{(l, r) | lur \in L\} = \{f | f \odot u \in L\}$

"Distributional Learning" models/exploits the distribution of strings;

Example

Simple CFL

$L = \{a^n b^n | n \geq 0\}$
$L = \{\lambda, ab, aabb \ldots\}$

Distribution of *aab*

- $(\lambda, b) \odot aab = aabb \in L$
- $(a, bb) \odot aab = aaabbb \in L$
- $(\lambda, abb) \odot aab = aababb \notin L$

Example

Simple CFL

$L = \{a^n b^n | n \geq 0\}$
$L = \{\lambda, ab, aabb \dots\}$

Distribution of *aab*

- $(\lambda, b) \odot aab = aabb \in L$
- $(a, bb) \odot aab = aaabbb \in L$
- $(\lambda, abb) \odot aab = aababb \notin L$

- $C_L(aab) = \{(\lambda, b), (a, bb), \dots (a^i, b^{i+1}) \dots\}$
- $C_L(aaabb) = C_L(aab)$
- $C_L(a) = \{(\lambda, b), (a, bb), (\lambda, abb) \dots\}$

Outline

Distributional learning

Congruence classes
- Syntactic monoid
- Canonical CFG

Algorithms
- Substitutable languages
- MAT learner
- NTS languages

Conclusion

Congruence classes and the syntactic monoid

Congruence classes

$u \equiv v$ iff $C_L(u) = C_L(v)$
Write $[u]$ for class of u

- Two strings are congruent if they are perfectly substitutable in every context.
- Words: Tuesday/Wednesday, cat/dog, man/student

Congruence classes of a regular language

Example $L = (ab)^*$

- $[\lambda] = \{\lambda\}$
- $[a] = \{a, aba, ababa, \dots\}$
- $[b] = \{b, bab, babab, \dots\}$
- $[ab] = \{ab, abab, \dots\}$
- $[ba] = \{ba, baba, \dots\}$
- $[bb]$ every other string with $C_L(u) = \emptyset$

Regular language

Theorem

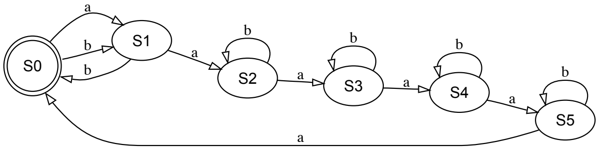
A regular languages has finitely many congruence classes.

Proof

Later.

Number of congruence classes may be exponential in size of minimal DFA: $|Q|^{|Q|}$.

Example blowup



Outline

Distributional learning

Congruence classes
- Syntactic monoid
- Canonical CFG

Algorithms
- Substitutable languages
- MAT learner
- NTS languages

Conclusion

Monoid

Simple algebraic structure

Definition

Associative operation $\circ$ with an element 1

$1 \circ u = u = u \circ 1$

Examples

Strings under concatenation

Numbers under addition

Matrices under multiplication

Syntactic monoid

Concatenation

If $u \equiv u'$ and $v \equiv v'$ then $uv \equiv u'v'$
$[u][v] \subseteq [uv]$

Example $L = (ab)^*$

$[b][ab] = \{b, bab, \dots\}\{ab, abab \dots\}$
$= \{bab, babab \dots\} \subset \{b, bab \dots\} = [b]$

Syntactic monoid

$\Sigma^* / \equiv_L$
$[u] \circ [v] = [uv]$
Associative and $[\lambda]$ is identity

Zero

Suppose $Sub(L)$ is not equal to Σ^*
There are strings that do not occur as a substring of any string in L.

Example

$L = \{(ab)^*\}$
$bb \notin Sub(L)$
$C_L(bb) = \emptyset$

Fact: if $u \notin Sub(L)$ then $uv \notin Sub(L)$
So we may have a congruence class $\mathbf{0}$; $\mathbf{0} \circ X = \mathbf{0} = X \circ \mathbf{0}$

Example

$L = \{(ab)^*\}$

$\circ$	$[\lambda]$	$[bb]$	$[a]$	$[b]$	$[ab]$	$[ba]$
$[\lambda]$	$[\lambda]$	$[bb]$	$[a]$	$[b]$	$[ab]$	$[ba]$
$[bb]$	$[bb]$	$[bb]$	$[bb]$	$[bb]$	$[bb]$	$[bb]$
$[a]$	$[a]$	$[bb]$	$[bb]$	$[ab]$	$[bb]$	$[a]$
$[b]$	$[b]$	$[bb]$	$[ba]$	$[bb]$	$[b]$	$[bb]$
$[ab]$	$[ab]$	$[bb]$	$[a]$	$[bb]$	$[ab]$	$[bb]$
$[ba]$	$[ba]$	$[bb]$	$[bb]$	$[b]$	$[bb]$	$[ba]$

Exercise

Language

$\{w | |w|_a = |w|_b\}$

Language of equal numbers of a's and b's

Question

What is the syntactic monoid of this?

What is it isomorphic too?

Outline

Distributional learning

Congruence classes
- Syntactic monoid
- Canonical CFG

Algorithms
- Substitutable languages
- MAT learner
- NTS languages

Conclusion

Context free grammar

Suppose we have a grammar with non-terminals N, P, Q

- We have a rule $N \rightarrow PQ$
- This means that $Y(N) \supseteq Y(P)Y(Q)$.

Context free grammar

Suppose we have a grammar with non-terminals N, P, Q

- We have a rule $N \rightarrow PQ$
- This means that $Y(N) \supseteq Y(P)Y(Q)$.

Backwards

Given a collection of sets of strings X, Y, Z
Suppose $X \supseteq YZ$
Then we add a rule $X \rightarrow YZ$.

Congruence classes

Partition of the strings

Congruence classes have nice properties!

$[u][v] \subseteq [uv]$
$[uv] \rightarrow [u][v]$
$[u] \circ [v] \rightarrow [u][v]$

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

$[a] = \{a\}$
$[abb] = \{abb, aabbb, \dots\}$
$[a][abb] = \{aabb, aaabbb \dots\} \subseteq [aabb] = [ab]$
So we have a rule $[ab] \rightarrow [a][aab]$

Constructing CFG from congruence classes

One non-terminal per congruence class

- $[uv] \rightarrow [u][v]$
- $[a] \rightarrow a$
- $[\lambda] \rightarrow \lambda$
- $I = \{[u] | [u] \subseteq L\}$

Multiple start symbols.

Example

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

Grammar

- $I = \{[ab], [\lambda]\}$

Example

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

Grammar

- $I = \{[ab], [\lambda]\}$
- $[a] \rightarrow a, [b] \rightarrow b, [\lambda] \rightarrow \lambda$

Example

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

Grammar

- $I = \{[ab], [\lambda]\}$
- $[a] \to a, [b] \to b, [\lambda] \to \lambda$
- $[ab] \to [aab][b], [ab] \to [a][b], [ab] \to [a][abb]$
- $[aab] \to [a][ab]$, $[abb] \to [ab][b]$

Example

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

Grammar

- $I = \{[ab], [\lambda]\}$
- $[a] \rightarrow a, [b] \rightarrow b, [\lambda] \rightarrow \lambda$
- $[ab] \rightarrow [aab][b], [ab] \rightarrow [a][b], [ab] \rightarrow [a][abb]$
- $[aab] \rightarrow [a][ab]$, $[abb] \rightarrow [ab][b]$
- Plus $[ba] \rightarrow [b][ba] \ldots$
- Plus $[a] \rightarrow [\lambda][a] \ldots$

Problems

If we can figure out whether $u \equiv_L v$ then we can write down a grammar.

Problem A

- In general we will have an infinite set of congruence classes
- Pick some finite subset of them
- This does not give us every CFL $\{a^n b^m | n < m\}$

Problem B

Hard to tell whether $u \equiv_L v$
We need some way of testing this.

Outline

Distributional learning

Congruence classes

- Syntactic monoid
- Canonical CFG

Algorithms

- Substitutable languages
- MAT learner
- NTS languages

Conclusion

Three algorithms

All based on the same representational idea:

1. Substitutable languages from positive data
2. Congruential languages with queries
3. NTS languages with stochastic positive examples

Outline

Two ideas

Chomsky review of Greenberg, 1959

let us say that two units A and B are substitutable$_1$ if there are expressions X and Y such that XAY and XBY are sentences of L.; substitutable$_2$ if whenever XAY is a sentence of L then so is XBY and whenever XBY is a sentence of L so is XAY (i.e. A and B are completely mutually substitutable). These are the simplest and most basic notions.

Problem:

we need substitutability$_2$ but what we observe is substitutability$_1$

Old concept

John Myhill, 1950 commenting on Bar-Hillel

I shall call a system *regular* if the following holds for all expressions μ, ν and all wffs ϕ, ψ each of which contains an occurrence of ν: If the result of writing μ for some occurrence of ν in ϕ is a wff, so is the result of writing μ for any occurrence of ν in ψ. Nearly all formal systems so far constructed are regular; ordinary word-languages are conspicuously not so.

Clark and Eyraud, 2005

A language is *substitutable* if $lur, lvr, l'ur' \in L$ means that $l'vr' \in L$.

substitutable and reversible

Clark and Eyraud, 2005

A language is *substitutable* if $lur, lvr, l'ur' \in L$ means that $l'vr' \in L$.

Angluin, 1982

A language is *reversible* if $ur, vr, ur' \in L$ means that $vr' \in L$.

A Bad Intuition

One context in common in enough

- The cat died
- The dog died

So “cat” and “dog” are congruent

A Bad Intuition

One context in common in enough

- The cat died
- The dog died

So “cat” and “dog” are congruent

- He is an Englishman
- He is thin

So “thin” and “an Englishman” are congruent.

Result

Clark and Eyraud, 2005/2007

Polynomial result

The class of substitutable context free languages is polynomially identifiable in the limit from positive data only.

- Polynomial characteristic set
- Polynomial update time

Why the delay?

A Simple Algorithm

Non-technical description

- Given a sample of strings $W = \{w_1, \ldots, w_n\}$.
- Define a graph $G = \langle N, E \rangle$
  - N is the set of all non-empty substrings (factors) of W.
  - $E = \{(u, v) | \exists (l, r), lur \in W \wedge lvr \in W\}$.
- Define a grammar in Chomsky normal form
  - The set of non-terminals is the set of components of the graph G
  - Have productions for: $[a] \rightarrow a$
  - Add rules for non terminals: $[w] \rightarrow [u][v]$ iff $[w] = [uv]$.

Simple linguistically motivated example

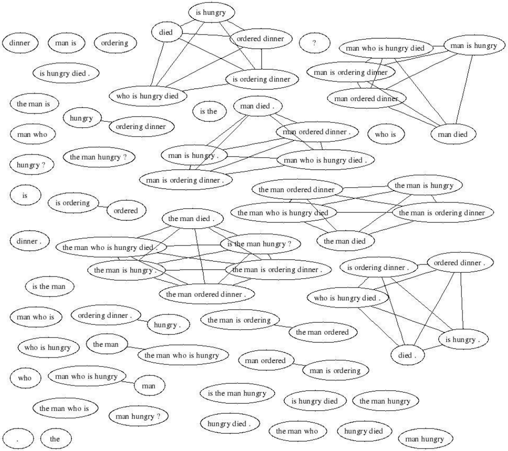
the man who is hungry died .
the man ordered dinner .
the man died .
the man is hungry .
is the man hungry ?
the man is ordering dinner .

---

is the man who is hungry ordering dinner ?
$*$is the man who hungry is ordering dinner ?

Substitution graph

Auxiliary fronting example

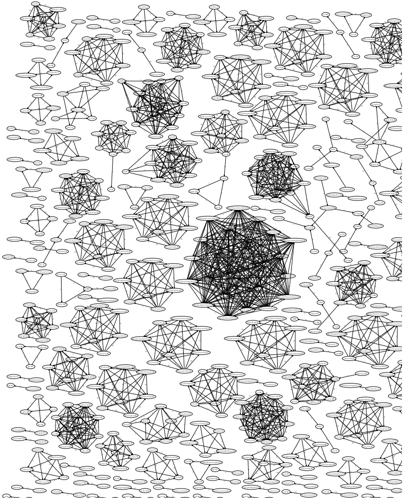


Derivations

[is the man hungry ?] $\rightarrow$ [is the man hungry] [?] $\rightarrow$ [is the man] [hungry] [?] $\rightarrow$ [is] [the man] [hungry] [?] $\rightarrow$ [is] [the man] [who is hungry] [hungry] [?] $\rightarrow$ [is] [the man] [who is hungry] [ordering dinner] [?].
Note that this will not work for all data since English is *not* substitutable (Berwick, Coen and Niyogi, p.c.)

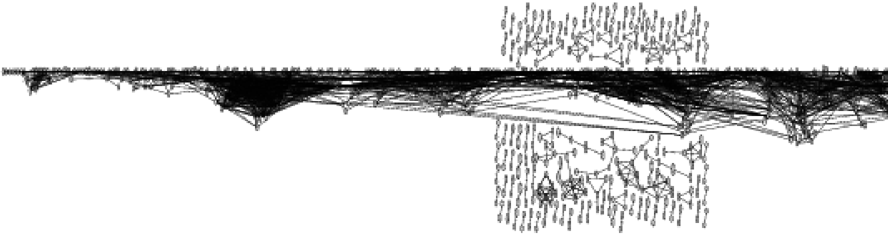
Substitution graph

Learnable example



Substitution graph

Unlearnable languages



Substitutable languages

- Some very basic languages are not substitutable:
  - $L = \{a, aa\}$
  - $L = \{a^n b^n | n > 0\}$
  - Dyck language
- The very strict requirement for contexts to be disjoint is unrealistic.
- If we move to a probabilistic learning approach, we can test for congruence with $|\hat{C}(u) - \hat{C}(v)|_\infty$.

Congruence class results

Positive data alone

$lur \in L$ and $lvr \in L$ implies $u \equiv_L v$
Polynomial result from positive data. (Clark and Eyraud, 2005)
k-l substitutable languages, (Yoshinaka 2008)

Stochastic data

If data is generated from a PCFG
PAC-learn unambiguous NTS languages, (Clark, 2006)

Membership queries

An efficient query-learning result (Clark, 2010)
Pick a finite set of contexts F
Test if $C_L(u) \cap F = C_L(v) \cap F$

Outline

ICGI 2010

Use query learning just as with LSTAR – the MAT model.

- Membership queries
- Equivalence queries – counterexamples

Efficient learning but not for all CFLs.

Algorithm

Maintain two sets:

- A set of strings K; includes Σ and λ
- A set of contexts F; includes (λ, λ)

- We have rows for K
- And also for KK – every pair.

Observation table

K a set of strings and F a set of contexts

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
k_8										
k_7										
k_6										
k_5										
k_4										
k_3										
k_2										
k_1										

Observation table

K a set of strings and *F* a set of contexts

	(λ, λ)	(a, λ)	(λ, b)
λ	■		
a			■
b		■	
ab	■		

Observation table

K a set of strings and F a set of contexts

	(λ,λ)	$(aaabb,bccc)$	$(aaabbc,\lambda)$	$(\lambda,abbccc)$	$(aaab,bccc)$	(aa,bbc)	$(aa,bbbc)$	(abb,cc)	$(abbb,cc)$
bcc									■
aab							■		
$bbcc$	■							■	
bc	■							■	
abc	■								
$aabb$	■					■			
ab	■					■			
c	■		■						■
b					■				
a	■			■			■		
λ	■	■				■		■	

Observation table

K a set of strings and F a set of contexts

	(aa, bbc)	(λ, λ)	(abb, cc)	$(aaabb, bccc)$	$(aaabbc, \lambda)$	$(\lambda, abbccc)$	$(aaab, bccc)$	$(aa, bbbc)$	$(abbb, cc)$
bcc									■
aab								■	
abc		■							
$aabb$	■	■							
ab	■	■							
λ	■	■	■	■					
$bbcc$		■	■						
bc		■	■						
c		■			■				■
b							■		
a		■				■		■	

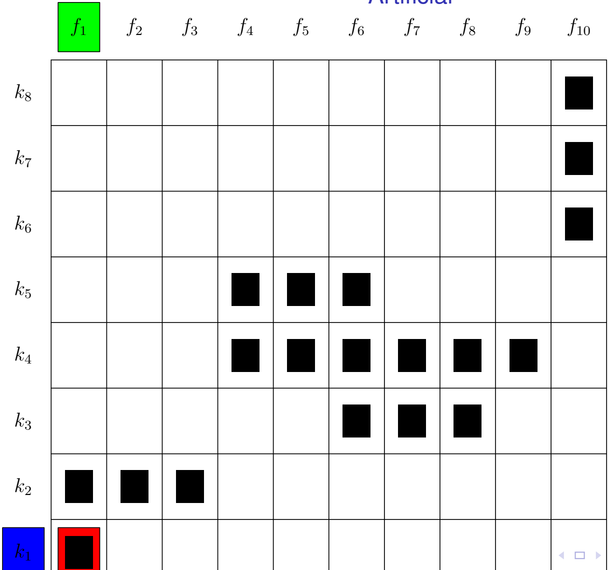
Substitutable

$L = \{a^n c b^n | n \geq 0\}$

	(λ,λ)	(a,b)	(λ,cb)	(ac,b)	(a,λ)	(λ,b)	(ac,λ)	(aac,b)
$aacb$						■		
ac						■		
$acbb$					■			
cb					■			
$aacbb$	■	■						
acb	■	■						
c	■	■						
b							■	■
a			■					

Example

Artificial



Example

Artificial

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
k_8										■
k_7										■
k_6										■
k_5				■	■	■				
k_4				■	■	■	■	■	■	
k_3						■	■	■		
k_2	■	■	■							
k_1	■									

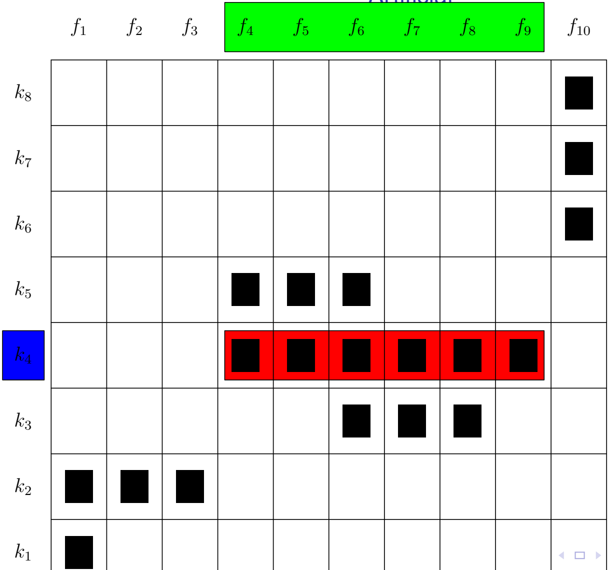
Example

Artificial

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
k_8										■
k_7										■
k_6										■
k_5				■	■	■				
k_4				■	■	■	■	■	■	
k_3						■	■	■		
k_2	■	■	■							
k_1	■									

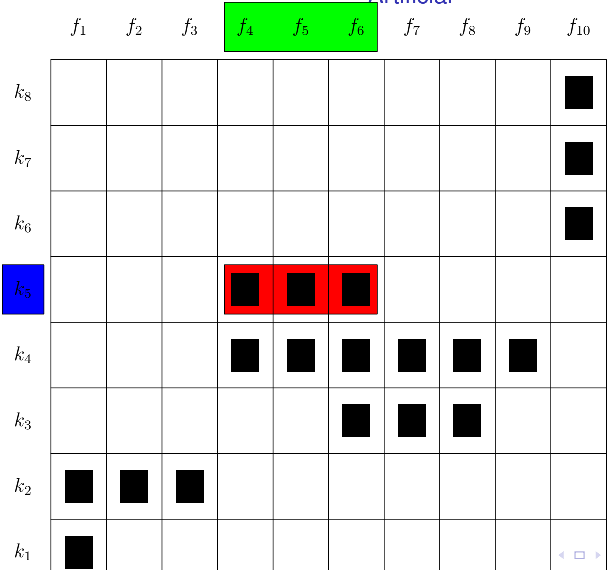
Example

Artificial



Example

Artificial



Example

Artificial

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
k_8										■
k_7										■
k_6										■
k_5				■	■	■				
k_4				■	■	■	■	■	■	
k_3						■	■	■		
k_2	■	■	■							
k_1	■									

Constructing CFG

Given the observation table:

- Non-terminals are equivalence classes of rows: $w \in N$ if N is $[w]$
- Add $N \rightarrow PQ$ if there is a $u \in P, v \in Q$ and $uv \in N$.
- Initial non-terminals are those rows with (λ, λ): $S \rightarrow N$ iff $N \subseteq L$

Example

Dyck language

	(λ, λ)	(a, λ)	(λ, b)
λ	1	0	0
a	0	0	1
b	0	1	0
ab	1	0	0
aab	0	0	1
abb	0	1	0
aa	0	0	0
ba	0	0	0
bb	0	0	0
bab	0	1	0
aba	0	0	1
$abab$	1	0	0

Example

Dyck language

	(λ, λ)	(a, λ)	(λ, b)
λ	1	0	0
ab	1	0	0
$abab$	1	0	0
a	0	0	1
aab	0	0	1
aba	0	0	1
b	0	1	0
abb	0	1	0
bab	0	1	0
aa	0	0	0
ba	0	0	0
bb	0	0	0

Discard rows with no element in K.

Example

Dyck language

Three classes

- $\{\lambda, ab, abab\}$ – Call this S
- $\{a, aab, aba\}$ – A
- $\{b, bab, abb\}$ – B.

Add rules

- $S \rightarrow AB$, $S \rightarrow SS$
- $A \rightarrow AS, SA, a$
- $B \rightarrow BS, SB, b$

Note that there is no rule $X \rightarrow AA$.

Closure

LSTAR

In the LSTAR algorithm, the table needed to be closed.
For every transition we needed to represent the state at the end.
For every $u \in K$, $a \in \Sigma$, we needed a $v \in K$ such that $(ua)^{-1}L = v^{-1}L$.

This algorithm: non-regular language

There are an infinite number of congruence classes
We cannot have a closed table

- Doesn't matter
- With a DFA – unique derivation
- With a CFG – lots of different trees

Consistency

The example is consistent:

- If $u \sim_F u'$ and $v \sim_F v'$, then $uv \sim_F u'v'$.
- If it is not consistent then we know something is wrong and we can make it consistent by adding a feature.
- If (l, r) is a context that distinguishes uv and $u'v'$
  - One of $(lu, r), (l, vr), (lu', r), (l, v'r)$ will split u, u' or v, v'.
- But it might take exponential time – each context could be twice the size of the previous one.

Not essential so leave it out.

Undergenerating

If we get a counter-example $w \in L$, then we add $Sub(w)$ to K.

Lemma

Each positive counter-example will give us at least one of the non-terminals in the target. Thus we will get at most n positive counterexamples for a CFG of size n

Overgeneralising

If we have enough contexts then we won't overgeneralise.
If we overgeneralise, we don't have enough contexts.

Problem

$S \overset{*}{\Rightarrow} w$, $S \in I$, but $w \notin L$.

Triple $\langle w, N, (l, r) \rangle$

- $N \overset{*}{\Rightarrow} w$
- All elements of N should have (l, r)
- $(l, r) \notin C_L(w)$

FindContext

Recursive

Goal: find a set of strings N and a context (l, r) that splits N.

- $N \stackrel{*}{\Rightarrow} w$
- $N \rightarrow PQ \stackrel{*}{\Rightarrow} uv = w$
- Assume all elements of N have the feature (l, r)
- So we must have $u'v' \in N$, $u' \in P$, $v' \in Q$.

Table

Possibilities

luvr NO

lu'vr luv'r

lu'v'r YES

- Query the two gaps

Table 1

luvr NO

lu'vr NO luv'r YES

lu'v'r YES

v cannot be congruent to v' as (lu', r) and $(lu, r) \in C_L(v')$ but not $C_L(v)$.

Table 2

luvr NO

lu'vr YES luv'r NO

lu'v'r YES

u cannot be congruent to u' as $(l, v'r)$ and $(l, vr) \in C_L(u')$ but not $C_L(u)$.

Table 3

luvr NO

lu’vr YES luv’r YES

lu’v’r YES

u cannot be congruent to u': (l, vr)

v cannot be congruent to v': (lu, r)

Table 4

luvr NO

lu’vr NO luv’r NO

lu’v’r YES

u cannot be congruent to u': $(l, v'r)$

v cannot be congruent to v': (lu', r)

Recursion

$\langle w, N, (l, r) \rangle$

- Start at root with (λ, λ)
- Go down through the parse tree:
- At each step we have a triple such that at least one element of N has the context (l, r), but w does not.
- Terminate when we can split N
- If we get down to a leaf we will always terminate, as $w \in N$

Algorithm

Result: A CFG G

$K \leftarrow \Sigma \cup \{\lambda\}$, $K_2 = K$;
$F \leftarrow \{(\lambda, \lambda)\}$;
$D = L \cap \{\lambda\}$;
$G = \langle K, D, F \rangle$;
while *true* **do**
    if `Equiv(`G`)` *returns correct* **then**
        return G ;
    $w \leftarrow$ `Equiv(`G`)` ;
    if *w is not in $L(G)$* **then**
        $K \leftarrow K \cup Sub(w)$;
    else
        $F \leftarrow$ `AddContexts(`G,w`)`;
    $G \leftarrow$ `MakeGrammar(`K, D, F`)` ;

Algorithm 1: LearnCFG

Example

Step 0	(λ, λ)
λ	1

Example

Step 1	(λ, λ)
λ	1
a	0
b	0
ab	1
aa	0
ba	0
bb	0
abb	0
aba	0
aab	0
bab	0
abab	1

Example

Step 2	(λ, λ)	(a, λ)
λ	1	0
a	0	0
b	0	1
ab	1	0
aa	0	0
ba	0	0
bb	0	0
abb	0	1
aba	0	0
aab	0	0
bab	0	1
abab	1	0

Example

Step 3	(λ, λ)	(a, λ)	(λ, b)
λ	1	0	0
a	0	0	1
b	0	1	0
ab	1	0	0
aa	0	0	0
ba	0	0	0
bb	0	0	0
abb	0	1	0
aba	0	0	1
aab	0	0	1
bab	0	1	0
abab	1	0	0

Result

Clark, ICGI 2010

Theorem

This algorithm polynomially learns the class of congruential context free languages from MQs and EQs

Language class

A CFG is congruential if $N \stackrel{*}{\Rightarrow} u, N \stackrel{*}{\Rightarrow} v$ implies $u \equiv_L v$

Observation

The same approach works for positive data and membership queries alone

Outline

Distributional learning

Congruence classes
- Syntactic monoid
- Canonical CFG

Algorithms
- Substitutable languages
- MAT learner
- NTS languages

Conclusion

NTS Languages

Boasson and Senizergues

Up to now we have relied on undecidable language theoretic properties. NTS is a decidable syntactic property.

Definition

A grammar G is non terminally separated (NTS) iff for every N, M if $N \overset{*}{\Rightarrow} \alpha$ and $M \overset{*}{\Rightarrow} u\alpha v$ then $M \overset{*}{\Rightarrow} uNv$.

NTS Languages

Boasson and Senizergues

Up to now we have relied on undecidable language theoretic properties. NTS is a decidable syntactic property.

Definition

A grammar G is non terminally separated (NTS) iff for every N, M if $N \stackrel{*}{\Rightarrow} \alpha$ and $M \stackrel{*}{\Rightarrow} u\alpha v$ then $M \stackrel{*}{\Rightarrow} uNv$.

For we like sheep have been led astray.

NTS Languages

Boasson and Senizergues

Up to now we have relied on undecidable language theoretic properties. NTS is a decidable syntactic property.

Definition

A grammar G is non terminally separated (NTS) iff for every N, M if $N \overset{*}{\Rightarrow} \alpha$ and $M \overset{*}{\Rightarrow} u\alpha v$ then $M \overset{*}{\Rightarrow} uNv$.

For we like sheep have been led astray.

Informally

If there is a string like “We like sheep” that can be a sentence, whenever you see an occurrence of “We like sheep” it can be an S.

Congruence classes of NTS languages

Congruence classes of non-terminals

Suppose G is an NTS grammar, and N is a non-terminal, then if $N \stackrel{*}{\Rightarrow} u$ and $N \stackrel{*}{\Rightarrow} v$ then $u \equiv_L v$.

Congruence classes of NTS languages

Congruence classes of non-terminals

Suppose G is an NTS grammar, and N is a non-terminal, then if $N \overset{*}{\Rightarrow} u$ and $N \overset{*}{\Rightarrow} v$ then $u \equiv_L v$.

- Thus we have a partial equivalence between the congruence classes of the language, and the non-terminals.
- Decidable property of CFG.
- Decidable equivalence

The MAT algorithm can learn all NTS languages.

PCFG

Suppose we have a PCFG which defines a probability distribution D.

Context distribution

$$C_D(u)[l, r] = \frac{P_D(lur)}{E_D(u)}$$

Note $\sum_{l,r} P_D(lur) = E_D(u)$, which could be greater than 1.

Unambiguous NTS languages

If we have an *unambiguous* NTS language then:

- Any two strings generated from the same non-terminal will have the same probabilistic distribution.

Ambiguous NTS languages

Some strings in $\{w | N \stackrel{*}{\Rightarrow} w\}$ could have different sets of derivations.
Add production $N \rightarrow w$ with large parameter.

PAC-learning Unambiguous NTS grammars

- If we restrict the distributions to those generated by a PCFG with the same structure then we can learn.

PAC-learning Unambiguous NTS grammars

- If we restrict the distributions to those generated by a PCFG with the same structure then we can learn.
- Parameters
    - μ_1-Distinguishability: same as with PDFAs

PAC-learning Unambiguous NTS grammars

- If we restrict the distributions to those generated by a PCFG with the same structure then we can learn.
- Parameters
  - μ_1-Distinguishability: same as with PDFAs
  - Separability: contexts are sufficiently far apart: A PCFG is ν-separable for some $\nu > 0$ if for every pair of strings u, v in $Sub(L(G))$ such that $u \not\equiv v$, it is the case that $L_\infty(C_u - C_v) \geq \nu \min(L_\infty(C_u), L_\infty(C_v))$

PAC-learning Unambiguous NTS grammars

- If we restrict the distributions to those generated by a PCFG with the same structure then we can learn.
- Parameters
  - μ_1-Distinguishability: same as with PDFAs
  - Separability: contexts are sufficiently far apart: A PCFG is ν-separable for some $\nu > 0$ if for every pair of strings u, v in $Sub(L(G))$ such that $u \neq v$, it is the case that $L_\infty(C_u - C_v) \geq \nu \min(L_\infty(C_u), L_\infty(C_v))$
  - Reachability: A PCFG is μ_2-reachable, if for every non-terminal $N \in V$ there is a string u such that $N \stackrel{*}{\Rightarrow}_G u$ and $L_\infty(C_u) > \mu_2$.

Result

Theorem

The class of unambiguous NTS grammars is PAC-learnable, with parameters μ_1, ν, μ_2, when the distributions are generated by a PCFG with the same structure as the NTS grammar.

Result

Theorem

The class of unambiguous NTS grammars is PAC-learnable, with parameters μ_1, ν, μ_2, when the distributions are generated by a PCFG with the same structure as the NTS grammar.

- Requirement for unambiguity is very strong with NTS grammars.
- Too many stratificational parameters?

Outline

Language class

Still limited

Includes

- All regular languages (syntactic monoid is finite)
- Dyck language
- $\{a^n b^n | n \geq 0\} \ldots$

Many simple languages are not in this class:

- Palindromes over $\{a, b\}$
- $L = \{a^n b^n | n \geq 0\} \cup \{a^n b^{2n} | n \geq 0\}$
- $L = \{a^n b^n c^m | n, m \geq 0\} \cup \{a^m b^n c^n | n, m \geq 0\}$

(Some subtle differences in classes but we conjecture that they define the same class)

Linguistic modelling

This seems close to the base model that is used in a lot of empirical work.

Limitations

Inadequate for natural language if Σ is a set of words:

- Exact substitutability is too strict
- Scholz and Pullum (2007): “fond” versus “proud”
- “cat” is not perfectly substitutable with “dog”
- Words are almost never perfectly substitutable; phrases sometimes.

The classes are

- far too small
- they are treated as completely unrelated

$$NP \rightarrow DT\ N$$

	company	companies	associate	furniture	sheep	oil
the	1	1	1	1	1	1
a	1				1	
an			1			1
this	1		1	1	1	1
those		1			1	
some		1		1	1	1

$NP \rightarrow DT\ N$

	company	companies	associate	furniture	sheep	oil
the	1	1	1	1	1	1
a	1				1	
an			1			1
this	1		1	1	1	1
those		1			1	
some		1		1	1	1

- Every determiner and noun is in a different congruence class
- Different rule for each combination

Regular language

Theorem

A regular languages has finitely many congruence classes.

Proof

Take a DFA for the language L with state set q
Define $f_w : Q \rightarrow Q$ such that $f_w(q) = \delta(q, w)$
There are finitely many functions and if $f_u = f_v$ then $u \equiv_L v$.

Number of congruence classes may be exponential in size of minimal DFA: $|Q|^{|Q|}$.

Monoid of regular languages

- $f_{uv} = f_u \circ f_v$
- If u has $\delta(q_1, u) = q_2$ and v has $\delta(q_2, v) = q_3$ then uv has $\delta(q_1, uv) = q_3$
- There is some structure in the congruence classes; we can view the function as being composed of simpler basis functions.

Summary

Context free

Congruence based approach

Define a set of primitives	Congruence classes
Derivation relation	Syntactic monoid
Language class	Subclass of CF languages
Inference algorithms	Testing congruence

We have made the change from regular to CF but results are too weak.