Three learnable models for the description of language

Alexander Clark

Department of Computer Science
Royal Holloway, University of London

25 May 2010
LATA, Trier

Learnable

THREE MODELS FOR THE DESCRIPTION OF LANGUAGE*

Noam Chomsky
rtment of Modern Languages and Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, Massachusetts

Outline

Methodological part

- Efficient learnability should be central to language theory
- Design representations to be intrinsically learnable
- Primitives based on objective features of the language

Technical part

Three specific models:

- Regular based on the residual languages
- Context-free based on the syntactic monoid
- Non context-free representations based on the Syntactic Concept Lattice

Outline

Language and Automata Theory and Applications
Two objects

Languages

Subsets of Σ^*
Distributions, $A^* \times B^*$, trees, ...
Class of languages $\mathcal{L}$

Finite Representations

Grammars, Automata, semi-Thue systems ...
Class of representations $\mathcal{G}$

What is the proper relation between these?

Typical definition

Normal direction

Define a representation G

Function from representation to language: $\mathcal{G} \to \mathcal{L}$

Non-terminal $\to$ set of strings derived from non-terminal

Context free grammar $G \to$ context free language $L(G)$

Fundamental computation

is $w \in L(G)$?

We want this to be efficient

Typical definition

Normal direction

Define a representation G
Function from representation to language: $\mathcal{G} \rightarrow \mathcal{L}$
Non-terminal $\rightarrow$ set of strings derived from non-terminal
Context free grammar $G \rightarrow$ context free language $L(G)$

Fundamental computation

is $w \in L(G)$?
We want this to be efficient

Where do these grammars come from?

Applications

Which of the two objects do we know about?

- Natural languages, DNA sequences ...
- We have information about the language.
- But we typically have no direct information about the representation.
  The representation may be a man-made object that we use to describe the language

(With one exception – programming and mark-up languages.)

Unsupervised Learning

Fundamental problem of linguistics

Language theory has its roots in linguistics:

Chomsky's questions (1986)

1. What constitutes knowledge of a language?
2. How is this knowledge acquired by its speakers?

Unsupervised Learning
Fundamental problem of linguistics

Language theory has its roots in linguistics:

Chomsky's questions (1986)

1. What constitutes knowledge of a language?
2. How is this knowledge acquired by its speakers?

- The fundamental problem of linguistics it to account for language acquisition
- We have no direct information about the representations that are used.
- The perceived hardness of this learning problem is "the existence proof for cognitive science" – Jerry Fodor.

PSGs were meant to be learnable

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.”

Go backwards

Put learnability first!

Opposite Direction

Given a language and we want to define a representation
Function from language to representation
$L \longrightarrow G(L)$
From set of strings $\longrightarrow$ representational primitive of formalism
Ideally $L(G(L)) = L$.

Fundamental computation

Construct representation $G(L)$ from L
We want this to be efficient

Two problems of grammar induction
First problem

Information theoretic problems

A general problem of learning:

- Absence of negative data (Gold, 1967)
- VC-dimension (Vapnik, 1998), covering numbers
- Sparsity, Noise etc.

We know how to attack these problems: MDL, NPB, MaxEnt

Not specific to grammatical inference

Two problems of grammar induction

Second problem

Computational problems

Complexity of finding a good hypothesis given this information

- Gold (1978), Kearns and Valiant (1989) ...
- Often based on embedding cryptographic problems in learning problems
- Specific to certain classes of representation

This is the crucial problem: given good information about the language, can we efficiently construct a representation?

Overview

	Inefficient	Efficient
Positive data and MQs	Gold (1967)	?
Stochastic data	Horning (1969) Angluin (1988) Chater and Vitanyi (2007)	?

These results suggest the presence of probabilistic data largely compensate for the absence of negative data. (Angluin, 1988)

Objective representations

Program

Given a language L

1. Define a collection of sets of strings as primitives
2. Define a derivation relation, based on algebraic properties of these sets.
3. Define a representation based on this derivation relation

Outline

Step 1

Make a representational decision

Residual languages

$u^{-1}L = \{w | uw \in L\}$
right congruence classes
$u \cong v$ iff $u^{-1}L = v^{-1}L$
$[u]_R = \{v | u \cong v\}$

Primitives

Finite set of strings K; $\lambda \in K$
$Q = \{[u]_R | u \in K\}$
$q_0 = [\lambda]_R$

Step 1

Make a representational decision

Residual languages

$u^{-1}L = \{w|uw \in L\}$
right congruence classes
$u \cong v$ iff $u^{-1}L = v^{-1}L$
$[u]_R = \{v|u \cong v\}$

Primitives

Finite set of strings K; $\lambda \in K$
$Q = \{[u]_R|u \in K\}$
$q_0 = [\lambda]_R$

Let's call the elements of Q "states"

Example

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

- $[\lambda]_R = L$
- $[a]_R = aL$
- $[b]_R = \{b, bb, bab, \ldots\}$ $(u^{-1}L = \emptyset)$

Define the goal

Basic property

If $u \in L$ then $[u]_R \subseteq L$
Let $F \subseteq Q$ be $\{[u] \in Q | u \in L\}$

- A representation defines a function from $\Sigma^* \to \{0, 1\}$
  $f(w) = 1$ iff $w \in L$
- Define a function from $\Sigma^* \to Q$
  We want $\delta(w) = q$ iff $w \in q$

Step 2
Recursive derivation

Algebraic property of the primitives

It is a right congruence:

$u \cong v \Rightarrow uw \cong vw$

$[u]_R \circ w = [uw]_R$

Derivation of δ

Left to right derivation:

- $\delta(\lambda) = [\lambda]_R = q_0$
- If $\delta(u) = q = [v]$ then $\delta(ua) = [va]_R$ if $[va]_R \in Q$

Language defined

If $\delta(w) = q$ and $q \in F$ then $w \in \hat{L}(K, L)$

DFA

A long route to a familiar destination

- This is just a deterministic automaton
- If $\delta(w) \in q$ then $w \in q$
- We will always undergenerate: $L \subseteq \hat{L}(K, L)$
- As K increases the language increases

Language class

If K is finite, then $\hat{L}(K, L)$ is regular
If L is regular and K is big enough then $\hat{L}(L, K) = L$
(Myhill-Nerode theorem)

Learning

Inference problems

How can we select K or Q?
How can we tell whether $u \cong v$?

Three solutions

1. restrict the class of languages
   $uw, vw \in L \Rightarrow u \cong v$ (Angluin, 1982)
2. pick a set of strings F
   $u^{-1}L \cap F = v^{-1}L \cap F$
   (Angluin, 1987)
3. Probabilistic measure of the similarity between $u^{-1}L$ and $v^{-1}L$, L_∞ norm
   (Ron et al. 1994), Clark and Thollard, 2004)

Summary of regular learning

Derived DFA starting from a representational assumption.

Minimal DFAs are learnable because there is a bijection between the representational primitives and some objectively defined sets of strings of the language.

Define a set of primitives	Right congruence classes
Derivation relation	Transition function
Language class	Regular languages
Inference algorithms	Testing right congruence

Outline

Distribution

Classic idea from structuralist linguistics:

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;

Classic distributional learning

Congruence classes

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

Syntactic monoid

$\Sigma^* / \equiv_L$
$[u] \circ [v] = [uv]$

Example $(ab)^*$

$[\lambda], [bb], [a], [b], [ab], [ba]$
Finitely many congruence classes iff L is regular.

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			■					

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 $N \to PQ$
- 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 $X \to YZ$.

Congruence classes
Partition of the strings

Congruence classes have nice properties!

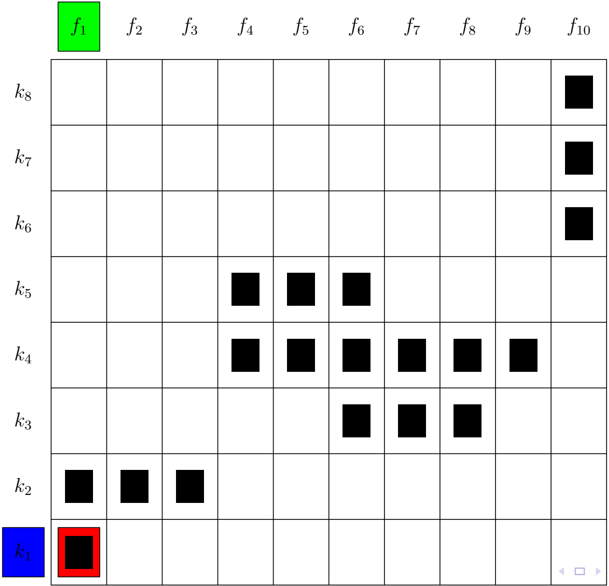
$[u][v] \subseteq [uv]$
$[uv] \rightarrow [u][v]$
$[u] \circ [v] \rightarrow [u][v]$

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

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

Example

Artificial

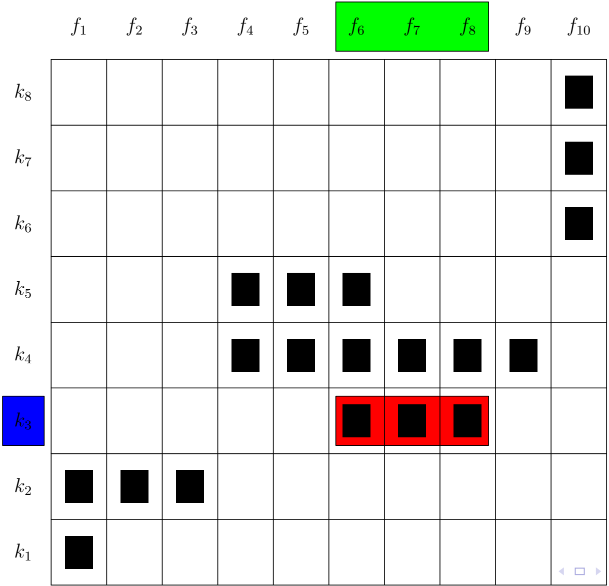


Example

Artificial

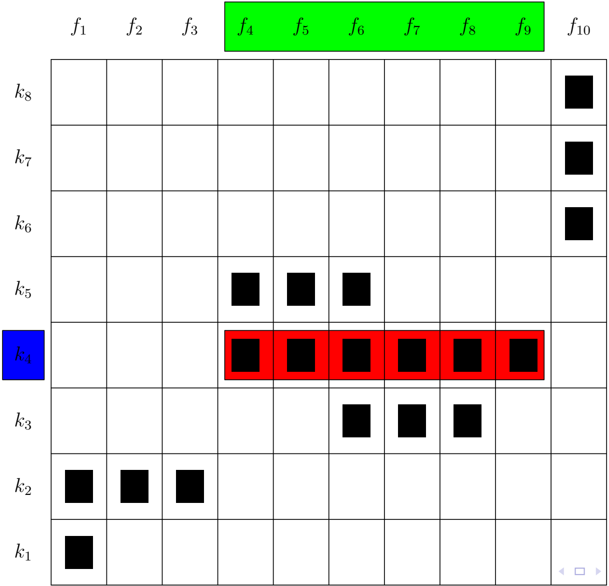
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
k_8										■
k_7										■
k_6										■
k_5				■	■	■				
k_4				■	■	■	■	■	■	
k_3						■	■	■		
k_2	■	■	■							
k_1	■									

Example
Artificial

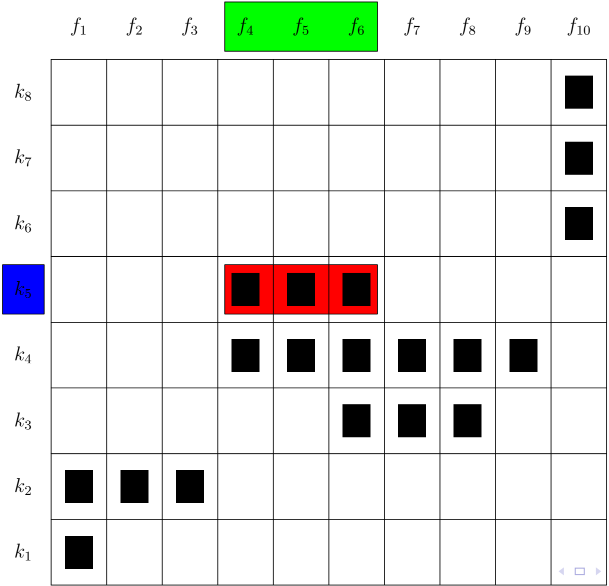


Example

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	■									

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

Grammar

- $S \rightarrow [ab], S \rightarrow [\lambda]$

Example

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

Grammar

- $S \rightarrow [ab], S \rightarrow [\lambda]$
- $[a] \rightarrow a, [b] \rightarrow b, [\lambda] \rightarrow \lambda$

Royal Holloway University of London

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

Grammar

- $S \rightarrow [ab], S \rightarrow [\lambda]$
- $[a] \rightarrow a, [b] \rightarrow b, [\lambda] \rightarrow \lambda$
- $[ab] \rightarrow [aab][b], [ab] \rightarrow [a][b], [ab] \rightarrow [a][abb]$
- $[aab] \rightarrow [a][ab]$, $[abb] \rightarrow [ab][b]$

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

Grammar

- $S \rightarrow [ab], S \rightarrow [\lambda]$
- $[a] \rightarrow a, [b] \rightarrow b, [\lambda] \rightarrow \lambda$
- $[ab] \rightarrow [aab][b], [ab] \rightarrow [a][b], [ab] \rightarrow [a][abb]$
- $[aab] \rightarrow [a][ab], [abb] \rightarrow [ab][b]$
- Plus $[ba] \rightarrow [b][ba] \dots$
- Plus $[a] \rightarrow [\lambda][a] \dots$

Problem

Hard to tell whether $u \equiv_L v$
If we can figure out the congruence classes, we can just write down a grammar.

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 (under submission)
Pick a finite set of contexts F
Test if $C_L(u) \cap F = C_L(v) \cap F$

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 CF languages are polynomially learnable.

Why the delay?

Language class
Quite limited

Class of CNF CFGs where each non-terminal generates a congruence class. (multiple S symbols)

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

Summary
Context free

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

Outline

Representational primitives

Strings

$[u] = \{v | v \equiv_L u\}$
The smallest possible sets
Partition

Contexts

$I[l, r] = \{v | lvr \in L\}$
These are the largest possible sets.
Overlap

Representational primitives

Strings

$[u] = \{v | v \equiv_L u\}$

The smallest possible sets

Partition

Contexts

$I[l, r] = \{v | lvr \in L\}$

These are the largest possible sets.

Overlap

Intersections

Given a set of contexts F

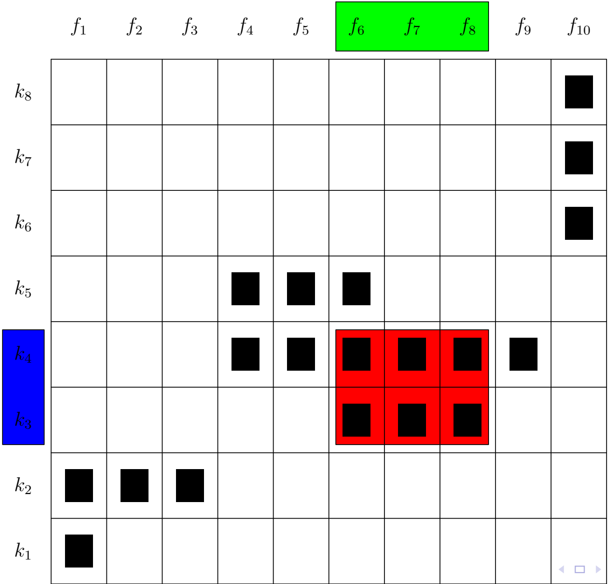
For every subset $C \subseteq F$

$C' = \{w | \forall (l, r) \in C, lwr \in L\}$

$\bigcap_{(l,r) \in C} I[l, r]$

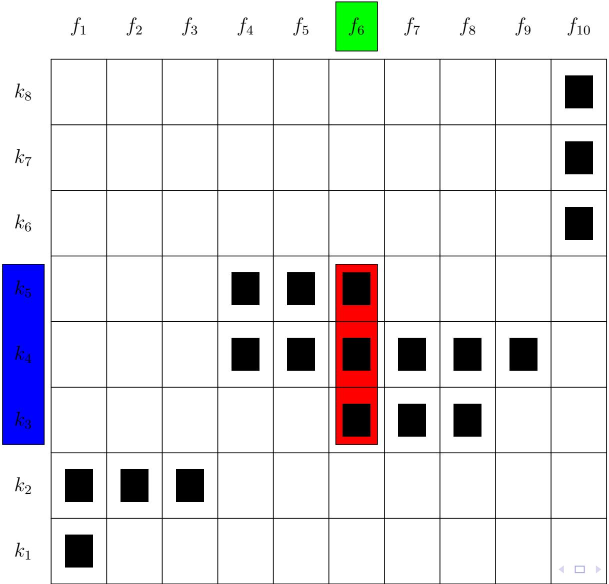
Concepts

Maximal rectangles

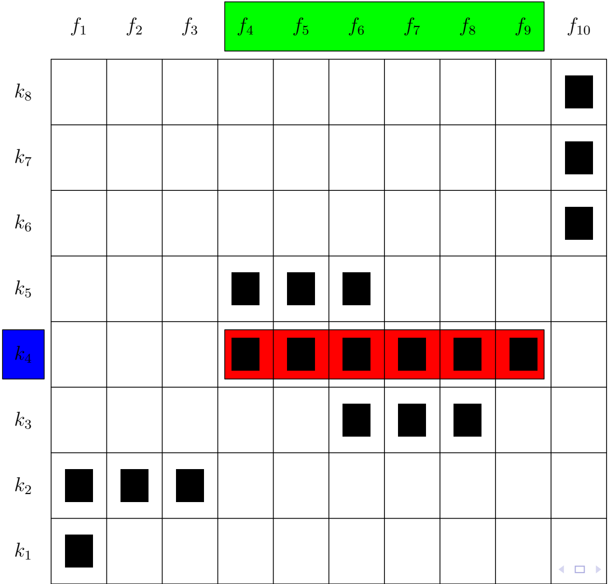


Concepts

Maximal rectangles

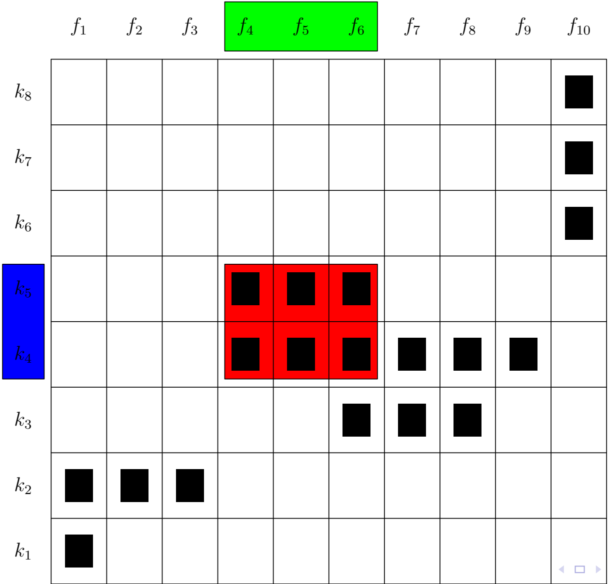


Concepts
Maximal rectangles

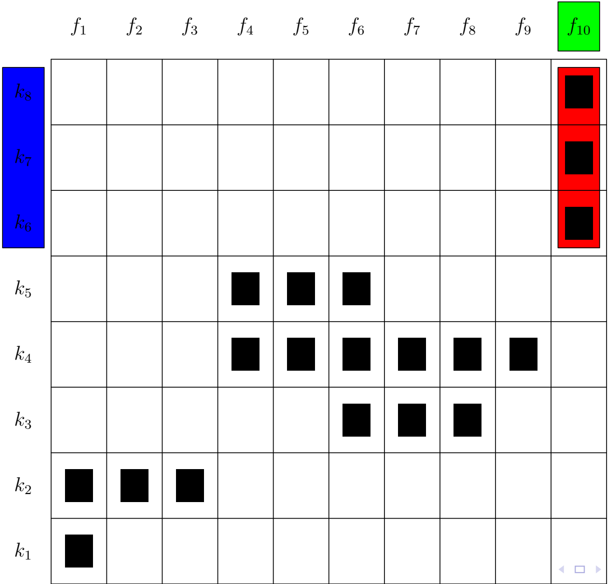


Concepts

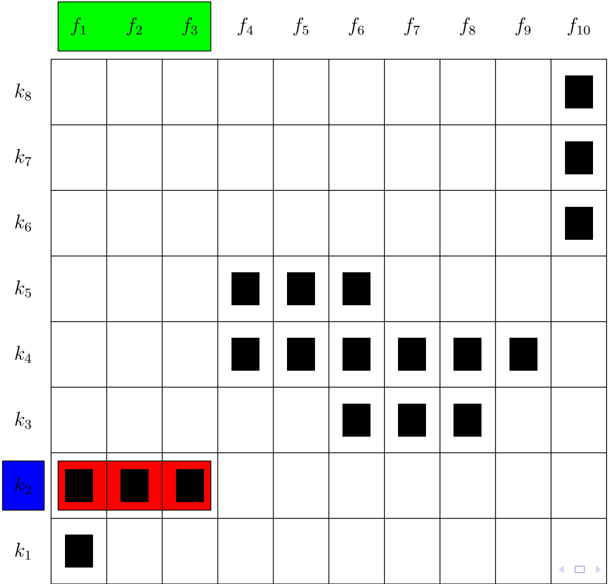
Maximal rectangles



Concepts
Maximal rectangles

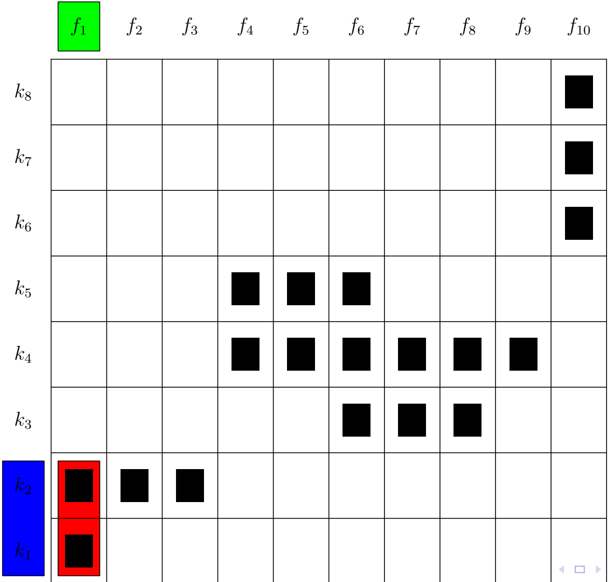


Concepts
Maximal rectangles

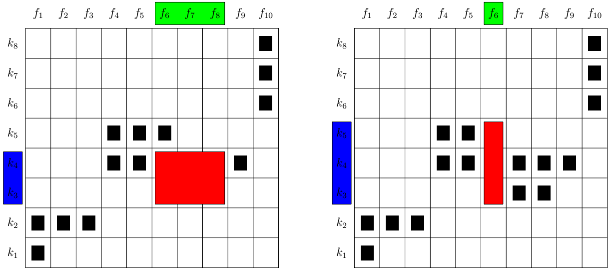


Concepts

Maximal rectangles



Partial order



Top and bottom

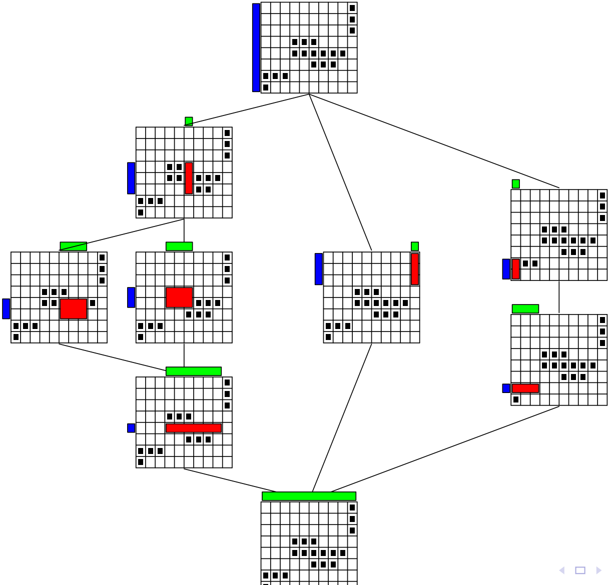
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
k_8										■
k_7										■
k_6										■
k_5				■	■	■				
k_4				■	■	■	■	■	■	
k_3						■	■	■		
k_2	■	■	■							
k_1	■									

Top and bottom

	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	■									

Complete Lattice

Formal Concept Analysis



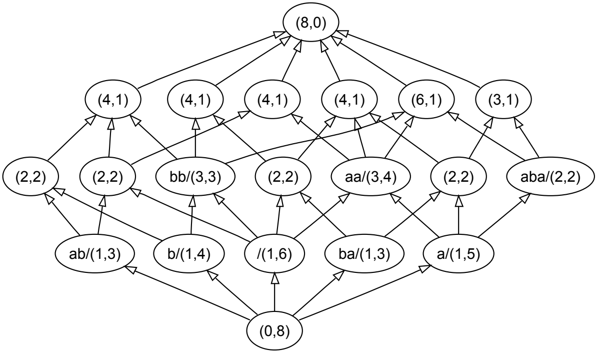
Lattice

Palindrome language over *a*, *b*

	(λ,λ)	(a,λ)	(ab,λ)	(aa,λ)	(b,λ)	(ba,λ)	(λ,b)	(λ,a)
aba	■		■					
bb	■				■		■	
ba		■	■				■	
ab					■	■		■
aa	■	■		■				■
b	■				■	■	■	
a	■	■	■	■				■

Lattice

Many rectangles



Formally

Polar maps

$$S' = \{(l, r) \in F : \forall w \in S \;\; lwr \in L\}$$
$$C' = \{w \in K : \forall (l, r) \in C \;\; lwr \in L\}$$

Concept

Ordered pair $\langle S, C \rangle$

- $S \subseteq K$ the set of strings
- $C \subseteq F$ is a set of contexts

$S' = C$ and $C' = S$

$\mathcal{C}(S) = \langle S'', S' \rangle$

Relation to CFGs

Define

Given a CFG G for each non-terminal N

- Yield: $Y(N) = \{w | N \stackrel{*}{\Rightarrow} w\}$
- Contexts: $C(N) = \{(l, r) | S \stackrel{*}{\Rightarrow} lNr\}$.

Clearly $C(N) \odot Y(N) \subseteq L$

Each non-terminal will be a rectangle – but not necessarily maximal.

Technical detail

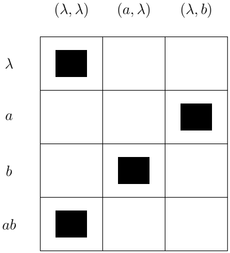
- These rectangles are "concepts" which form a complete lattice $\mathfrak{B}(K, L, F)$
- We use a concatenation operation $X \circ Y$ and a lower bound $X \wedge Y$.

Concatenation

$\langle S_1, C_1 \rangle \circ \langle S_2, C_2 \rangle = \langle (S_1 S_2)'', (S_1 S_2)''' \rangle$
Given two sets of strings S_x, S_y
Concatenate them $S_x S_y$
Find the shared set of contexts C_{xy}
Result is the concept defined by C'_{xy}

Dyck language

λ, *ab*, *abab*, *aabb*, *abaabb* . . .



- $L = \langle \{\lambda, ab\}, (\lambda, \lambda) \rangle$
- $A = \langle \{a\}, (\lambda, b) \rangle$
- $B = \langle \{b\}, (a, \lambda) \rangle$
- $\top$
- $\bot$

Dyck language

$\lambda, ab, abab, aabb, abaabb \ldots$

- $L = \langle \{\lambda, ab\}, (\lambda, \lambda) \rangle$
- $A = \langle \{a\}, (\lambda, b) \rangle$
- $B = \langle \{b\}, (a, \lambda) \rangle$
- $\top$
- $\bot$

	$\top$	L	A	B	$\bot$
$\top$	$\top$	$\top$	$\top$	$\top$	$\bot$
L	$\top$	L	A	B	$\bot$
A	$\top$	A	$\top$	L	$\bot$
B	$\top$	B	$\top$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

Goal

Predict which concept a string is in:

- Define function $\phi : \Sigma^* \to \mathfrak{B}(K, L, F)$
- A string w is in the language if $\phi(w)$ has the context (λ, λ).
- We want $\phi(w) = \langle S, C \rangle$ to mean that $C_L(w) \cap F = C$.

Recursive definition

- $\phi(a) = \mathcal{C}(a)$ (look it up)
- $\phi(ab) = \phi(a) \circ \phi(b)$
- $\phi(abc) = \phi(ab) \circ \phi(c)$, OR $\phi(a) \circ \phi(bc)$

Goal

Predict which concept a string is in:

- Define function $\phi : \Sigma^* \rightarrow \mathfrak{B}(K, L, F)$
- A string w is in the language if $\phi(w)$ has the context (λ, λ).
- We want $\phi(w) = \langle S, C \rangle$ to mean that $C_L(w) \cap F = C$.

Recursive definition

- $\phi(a) = \mathcal{C}(a)$ (look it up)
- $\phi(ab) = \phi(a) \circ \phi(b)$
- $\phi(abc) = \phi(ab) \circ \phi(c)$, OR $\phi(a) \circ \phi(bc)$
- $\phi(abc) = \phi(ab) \circ \phi(c) \wedge \phi(a) \circ \phi(bc)$

Distributional lattice grammars

Derivation: efficient $\mathcal{O}(|w|^3)$ algorithm

Definition

A distributional lattice grammar (DLG) is a tuple $\langle K, D, F\rangle$

- K is a finite subset of strings that includes Σ and λ
- F is a finite set of contexts that includes (λ, λ)
- D is a finite subset of $F \odot KK$

Definition

$\phi : \Sigma^* \to \mathfrak{B}(K, D, F)$.

- for all $a \in \Sigma$, $\phi(a) = \mathcal{C}(a)$
- for all w with $|w| > 1$,

$$\phi(w) = \bigwedge_{u,v \in \Sigma^+ : uv = w} \phi(u) \circ \phi(v)$$

Derivation example

Dyck language

$\{\lambda, ab, aabb, abab, aaababbb\ldots\}$
$F = \{(\lambda, \lambda), (\lambda, b), (a, \lambda)\}$
$K = \{\lambda, a, b, ab\}$

- $\top = \langle K, \emptyset \rangle$
- $\perp = \langle \emptyset, F \rangle$
- $\mathbf{L} = \langle \{\lambda, ab\}, \{(\lambda, \lambda)\} \rangle$
- $A = \langle \{a\}, \{(\lambda, b)\} \rangle$
- $B = \langle \{b\}, \{(a, \lambda)\} \rangle$

Derivation example

Dyck language

$\{\lambda, ab, aabb, abab, aaababbb\ldots\}$
$F = \{(\lambda,\lambda),(\lambda,b),(a,\lambda)\}$
$K = \{\lambda, a, b, ab\}$

1. $\phi(a) = A$
2. $\phi(ab) = \phi(a)\circ\phi(b) = \mathbf{L}$, $\phi(aa) = \top$ **...**
3. $\phi(aab) = \phi(a)\circ\phi(ab) \wedge \phi(aa)\circ\phi(b) = A$
4. **...**
5. $\phi(aaababbb) = \phi(a)\circ\phi(aababbb) \wedge \cdots = \mathbf{L}$

Example

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

	(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	■			■			■		

Example

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

	(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							■		

Example

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

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

Example

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

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

Learnability I

Search

Language is defined by choice of K and F
How can we find suitable K and F?

Lemma 1

as we increase K the language defined by $\langle K, L, F\rangle$ decreases monotonically
It will always converge to a subset of L in a finite time

Lemma 2

As we increase the set of contexts F the language monotonically increases.
Any sufficiently large set of contexts will do.

Search problem is trivial

Naive Algorithm

Start with $F = \{(\lambda, \lambda)\}, K = \Sigma \cup \{\lambda\}$

- If we see a string that is not in our hypothesis, the hypothesis is too small, and we add contexts to F
- Add strings to K if it will change the lattice at all.

Clark, (CoNLL, 2010)

DLGs can be learnt from positive data and MQs
Polynomial update time

Power of Representation

Language class

Let $\mathcal{L}$ be the set of all languages L such that there is a *finite* set of contexts F s.t. $L = L(\mathfrak{B}(\Sigma^*, L, F))$

Learnable class includes

1. All regular languages
2. Some but not all CFLs (all the examples so far)
3. Some non context free languages

Not in DLG?

$L = \{a^n b | n > 0\} \cup \{a^n c^m | m > n > 0\}$

	(a, λ)	(aa, λ)	(aaa, λ)	(a^4, λ)	(a^5, λ)	(a^6, λ)	(a^7, λ)
c^9	■	■	■	■	■	■	■
c^8	■	■	■	■	■	■	■
c^7	■	■	■	■	■	■	■
c^6	■	■	■	■	■	■	■
c^5	■	■	■	■	■	■	
c^4	■	■	■	■	■		
ccc	■	■	■	■			
cc	■	■	■				
c	■	■					
b	■	■	■	■	■	■	■

Context sensitive example

Let $M = \{(a, b, c)^*\}$, we consider the language
$L = L_{abc} \cup L_{ab} \cup L_{ac}$ where $L_{ab} = \{wd|w \in M, |w|_a = |w|_b\}$,
$L_{ac} = \{we|w \in M, |w|_a = |w|_c\}$,
$L_{abc} = \{wf|w \in M, |w|_a = |w|_b = |w|_c\}$.

$F = \{(\lambda, \lambda), (\lambda, d), (\lambda, ad), (\lambda, bd), (\lambda, e), (\lambda, ae), (\lambda, ce), (\lambda, f), (ab, \lambda),$

This is in the learnable class.

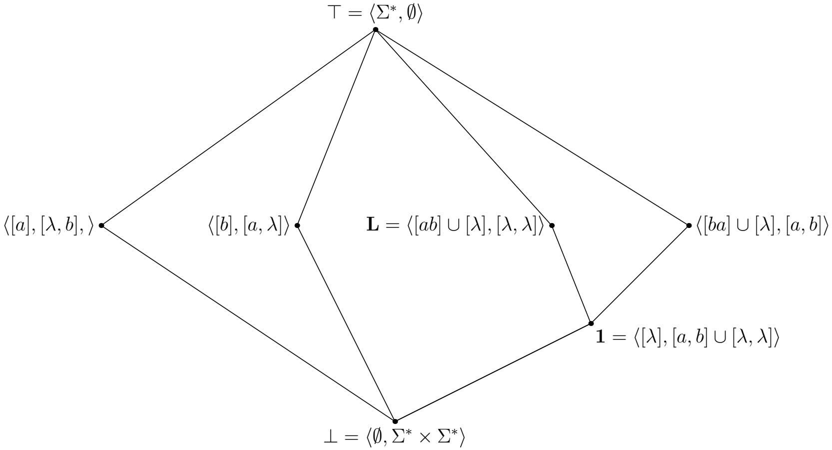
Syntactic concept lattice

Infinite limit

Let $K \to \Sigma^*$ and $F \to \Sigma^* \times \Sigma^*$
$\mathfrak{B}(K, L, F) \to \mathfrak{B}(L)$

- $\mathfrak{B}(L)$ is the *syntactic concept lattice*
- Same construction as the Universal Automaton for regular languages
- $\mathfrak{B}(L)$ is finite iff L is regular

$L = (ab)^*$



$\top = \langle \Sigma^*, \emptyset \rangle$
$\langle [a], [\lambda, b], \rangle$
$\langle [b], [a, \lambda] \rangle$
$\mathbf{L} = \langle [ab] \cup [\lambda], [\lambda, \lambda] \rangle$
$\langle [ba] \cup [\lambda], [a, b] \rangle$
$\mathbf{1} = \langle [\lambda], [a, b] \cup [\lambda, \lambda] \rangle$
$\bot = \langle \emptyset, \Sigma^* \times \Sigma^* \rangle$

Complete residuated lattice

- $X = \langle S_x, C_x \rangle$ and $Y = \langle S_y, C_y \rangle$ are concepts.
- Then define the residual $X / Y = \mathcal{C}(C_x \odot (\lambda, S_y))$
- $Y \setminus X = \mathcal{C}(C_x \odot (S_y, \lambda))$

These are unique, and satisfy the following conditions:

Lemma

$Y \leq X \setminus Z$ *iff* $X \circ Y \leq Z$ *iff* $X \leq Z / Y$.

Useful for displaced constituents in natural language: "the book that I told you to read"

Outline

General learning principle

Applies to other language classes:

- Finite languages – $\{w\}$
- k-locally testable languages
- Planar languages (Clark, Costa Florencio and Watkins, 2006)

The Future

Various relations

- $u \sim v$ iff $uv \in L$
- $(l, r) \sim u$ iff $lur \in L$

The Future

Various relations

- $u \sim v$ iff $uv \in L$
- $(l, r) \sim u$ iff $lur \in L$
- $(l, m, r) \sim (u, v)$ iff $lumvr \in L$ (Yoshinaka, 2009)

- Equational theory and relations to CG and pregroup grammars
- Decidability
- Better understanding of the class of languages

Learnability

- Grammatical inference is crucial and representation classes need to be designed to be learnable
- The structure of the representation should be based on the structure of the data
- Applying this approach gives efficient algorithms
  - Congruence based approaches using CFGs
  - Syntactic concept lattice
  - Distributional Lattice Grammars
- These are the only efficient algorithms for large classes of context free languages.