Computational Learning Theory
Extending Patterns with the Correctness of Learning Algorithms

Akihiro Yamamoto 山本 章博
http://www.iip.ist.i.kyoto-u.ac.jp/member/akihiro/
akihiro@i.kyoto-u.ac.jp

Contents

- What about a pair of patterns?
- Correctness of Learning Algorithms

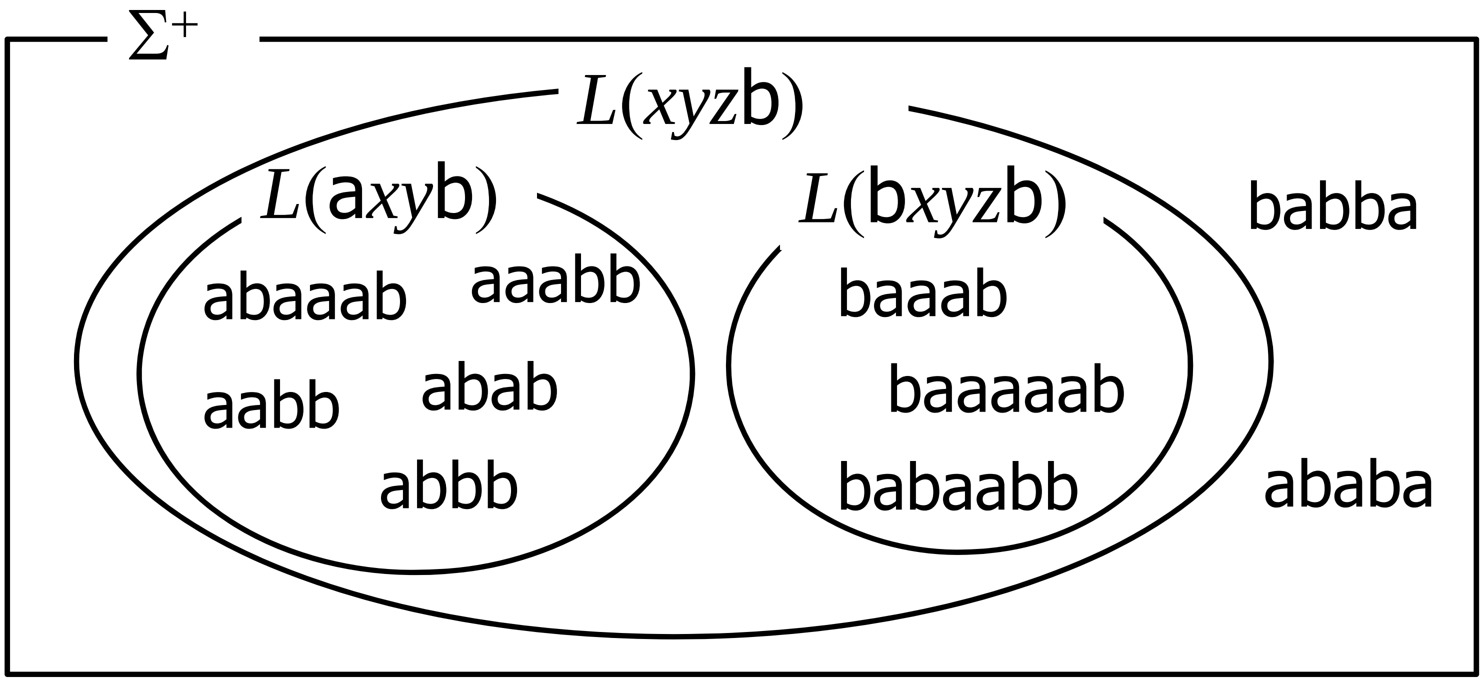
What about a pair of patterns?

Outputting a Pair of Patterns

$C = \{$babaabb, aaab, baaab, aabb, abab, baaaaab, abbb, aaabb, baaab$\}$

abaaab, aabb, abab, abbb, aaabb $\in L(\text{a}x\text{b})$

baaab, baaaaab, babaabb $\in L(\text{b}xyz\text{b})$

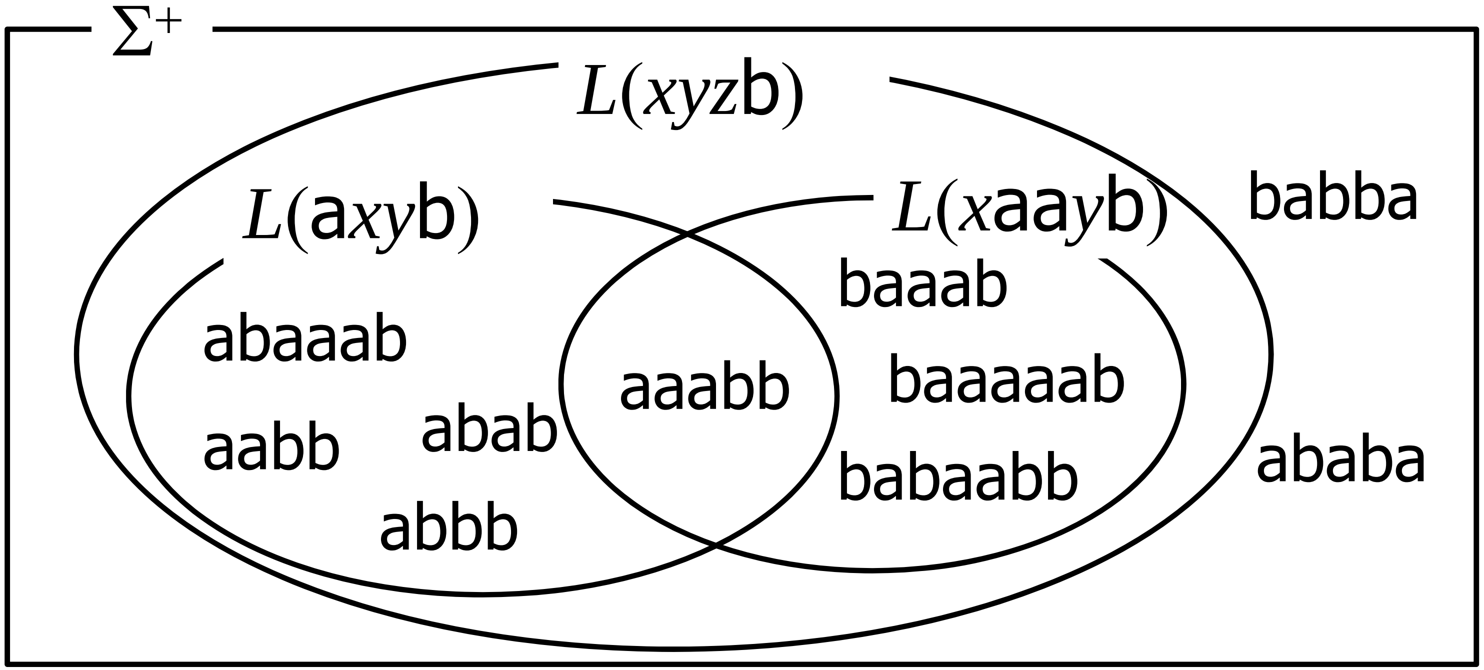


Outputting a Pair of Patterns

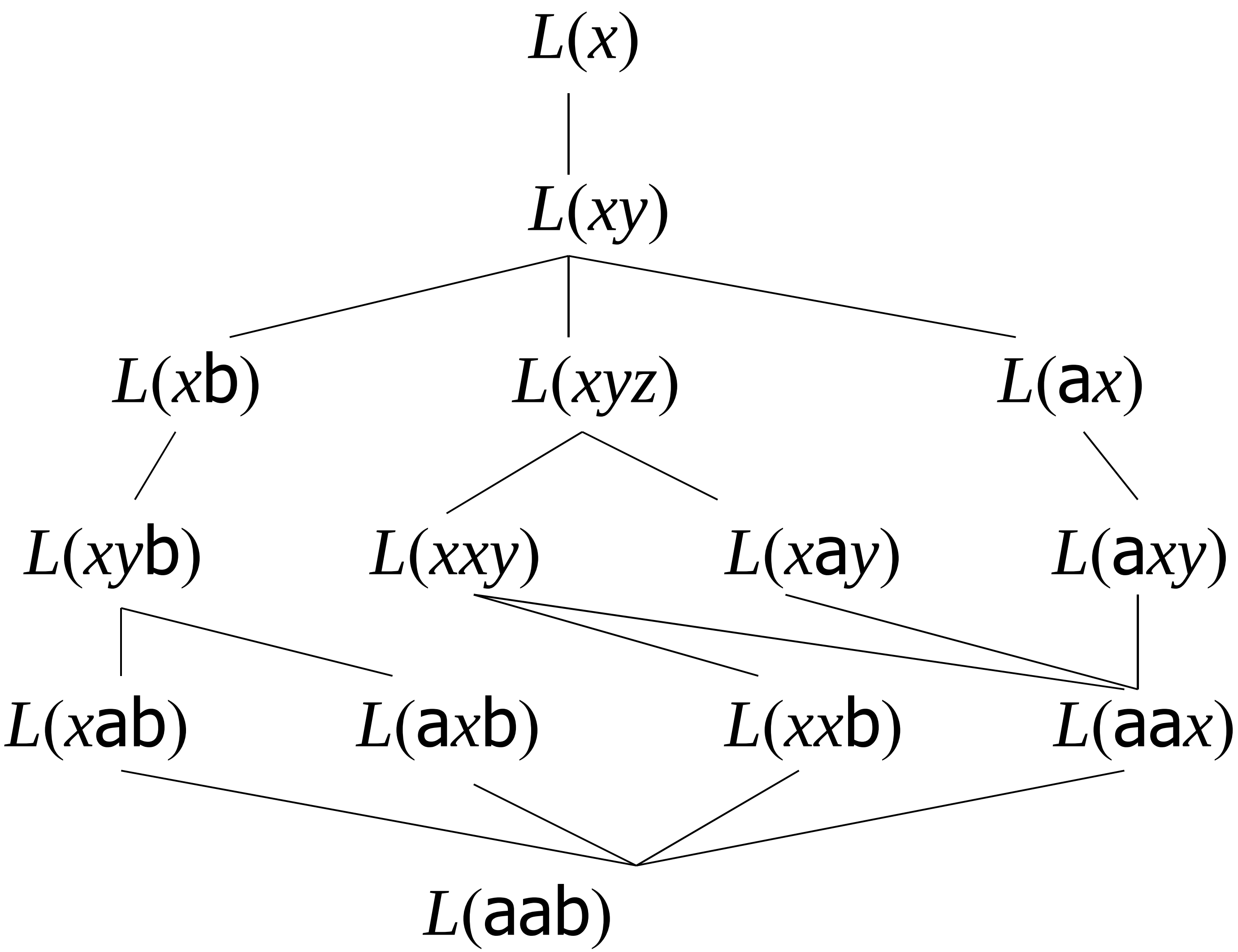
$C = \{$babaabb, aaab, baaab, aabb, abab, baaaaab, abbb, aaabb, baaab$\}$

abaaab, aabb, abab, abbb, aaabb $\in L($axyb$)$

baaab, baaaaab, aaabb, babaabb $\in L(x$aa$y$b$)$



Hasse Diagram



How to Construct H.D.

- A string w is an instance of π if $w = \pi\theta$ for some substitution θ.
- A pattern τ is one step refinement of π if $\tau = \pi\theta$ and θ is one of the three:
  - $\theta = \{(x, c)\}$ Replacing a variable with a symbol (character)
  - $\theta = \{(x, x_1x_2)\}$ Spritting a variable with two new variables
  - $\theta = \{(x, z), (y, z)\}$ Unifying two variables into one

Example

$$x \to x_1x_2 \to x_1x_2\,x_3 \to x_1x_1\,x_3 \to x_1x_1\mathrm{b}$$

Substitution (1)

- A substitution is a set of pairs

$$\theta = \{ (x_1, \tau_1), (x_2, \tau_2), \ldots, (x_n, \tau_n) \}$$

where $x_1, x_2, \ldots, x_n$ are distinct variables and

$\pi_1, \pi_2, \ldots, \pi_n$ are patterns.

- Applying a substitution θ to a pattern π is replacing every variable x_i in π with τ_i simultaneously.

The result is denoted by $\pi\theta$.

Example

$\theta_1 = \{ (x, \mathrm{bba}), (y, \mathrm{ba}) \}$

$\theta_2 = \{ (x, \mathrm{b}y\mathrm{a}), (y, \mathrm{a}y\mathrm{b}) \}$

$\mathrm{b}x\mathrm{a}x\mathrm{b}\theta_1 = \mathrm{bbbaabbab}$, $\mathrm{b}x\mathrm{a}x\mathrm{b}\theta_2 = \mathrm{bb}y\mathrm{aab}y\mathrm{ab}$,

$\mathrm{a}x\mathrm{bb}y\mathrm{a}\theta_1 = \mathrm{abbabbbaa}$, $\mathrm{a}x\mathrm{bb}y\mathrm{a}\theta_2 = \mathrm{ab}y\mathrm{abba}y\mathrm{ba}$

Anti-Unification

- A pattern π is an anti-unifier of the set C of positive examples if $C \subset L(\pi)$, i.e., for each positive example w, w=$\pi\theta$.
- A pattern π is a least common anti-unifier of C if π is an anti-unifier of C and no $\pi' \leq \pi$ satisfies $C \subset L(\pi')$.

Examples

The least common anti-uinifer of abaaab and aaabb is a*x*a*y*b.

The least common anti-uinifer of konnichiwa and konbanwa are kon*x*wa.

The least common anti-uinifers of konnichiwa and konbannwa are kon*x*wa and ko*x*n*y*wa.

Notes

- The operation of “making one step refinement” can be regarded as “applying derivative”

The learning algorithm *learn-patterns*

For $n = 1$ forever

receive $e_n = \langle s_n , b_n \rangle$

compute the list $l = \pi_1, \pi_2 , \ldots, \pi_k$ of all the least common anti-unifications of the set of positives $C_n = \{s_j : \langle s_j , + \rangle$ and $j = 1, 2, \ldots, n\}$

for each π_j in the list l

if an $e_j = \langle s_j , - \rangle$ s.t. $s_j \in L(\pi_j)$ is found

delete π_j from l

if l is not empty

return one π_i

In Theoretical Form

Lemma 2 Let $\pi_1, \pi_2, \ldots, \pi_n$ be patterns. If the language $L(\pi_k)$ is minimal in $\{L(\pi_1), L(\pi_2), \ldots, L(\pi_n)\}$, then π_k is one of the longest patterns in the list.

Lemma 3 Let π_1 and π_2 be patterns of same length. Then $L(\pi_1) \subseteq L(\pi_2)$ if and only if $\pi_2\theta = \pi_1$.

Note If we do not assume π_1 and π_2 be patterns of same length, then it is not decidable whether or not

$L(\pi_1) \subseteq L(\pi_2)$.

Which pattern should be chosen?

- Let C be a set of (positive) examples

1. Select all shortest examples.
2. Look for one of the minimal patterns between x (a singleton variable) and the anti-unifier of the shortest examples, and return it.

Note: If we only follow the identification-in-the-limit criterion, the second can be simplified as

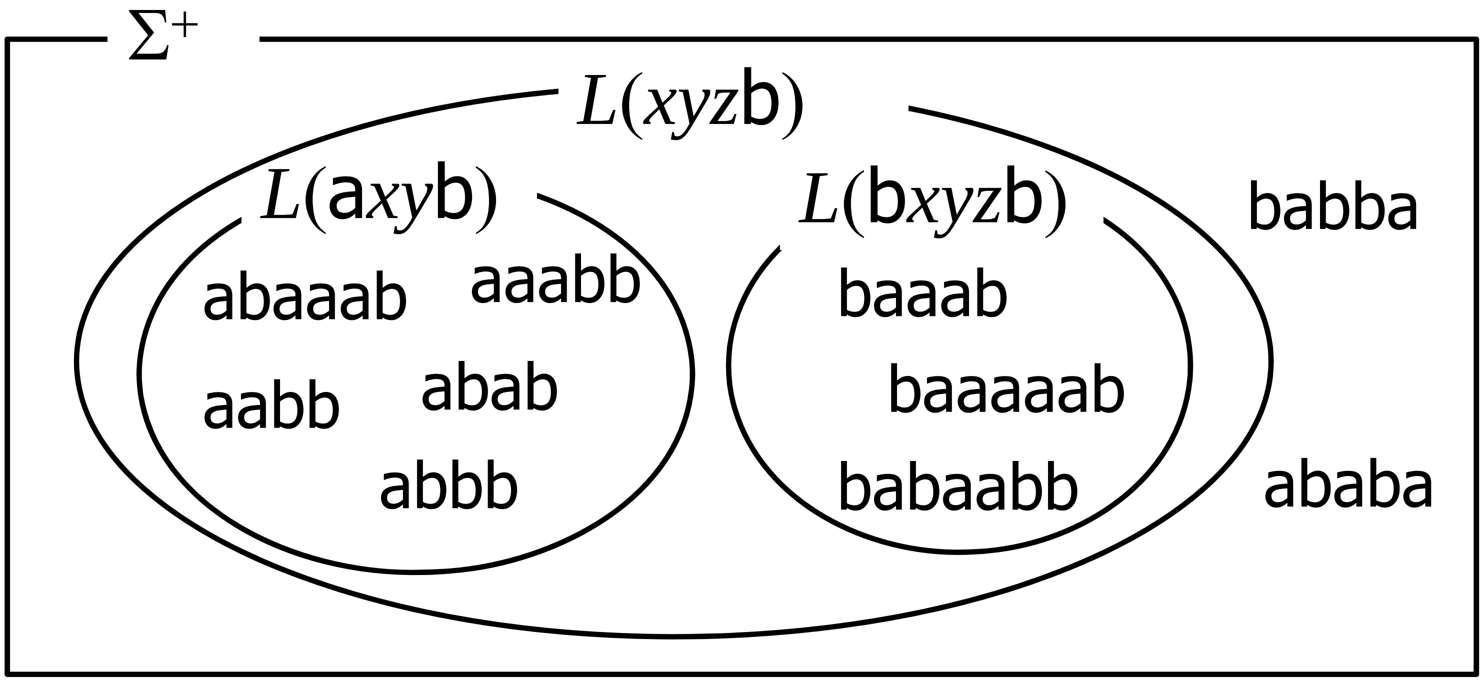
2'. Return the anti-unifier of the shortest examples but this might not seem "learning".

Outputting a Pair of Patterns

$C = \{$babaabb, aaab, baaab, aabb, abab, baaaaab, abbb, aaabb, baaab$\}$

abaaab, aabb, abab, abbb, aaabb $\in L(\mathrm{a}x\mathrm{b})$

baaab, baaaaab, babaabb $\in L(\mathrm{b}xyz\mathrm{b})$

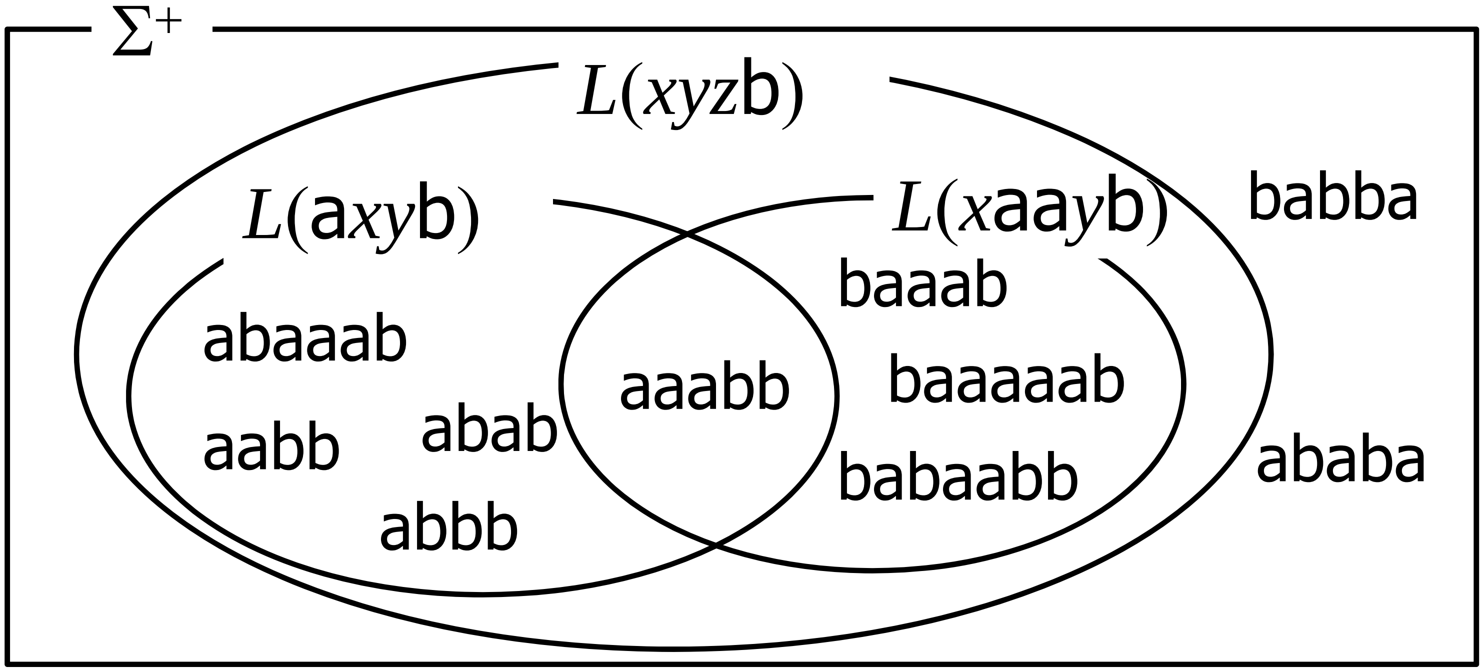


Outputting a Pair of Patterns

$C = \{$babaabb, aaab, baaab, aabb, abab, baaaaab, abbb, aaabb, baaab$\}$

abaaab, aabb, abab, abbb, aaabb $\in L(\mathrm{a}xy\mathrm{b})$

baaab, baaaaab, aaabb, babaabb $\in L(x\mathrm{aa}y\mathrm{b})$



Downward Coverset Algorithm

Given a data set C

For each minimally common anti-unifier π of C

For each a pair π_1, π_2 of patterns just beneth π in the Hasse Diagram

if $L(\pi_2) \subset C - L(\pi_1)$ then

make minimally common anti-unification π_1'

of $C \cap L(\pi_1)$

and

minimally common anti-unification π_2'of $C - L(\pi_1)$

Correctness of Learning Algorithms

Positive and Negative examples

 $e_1, e_2, e_3, \ldots$ 

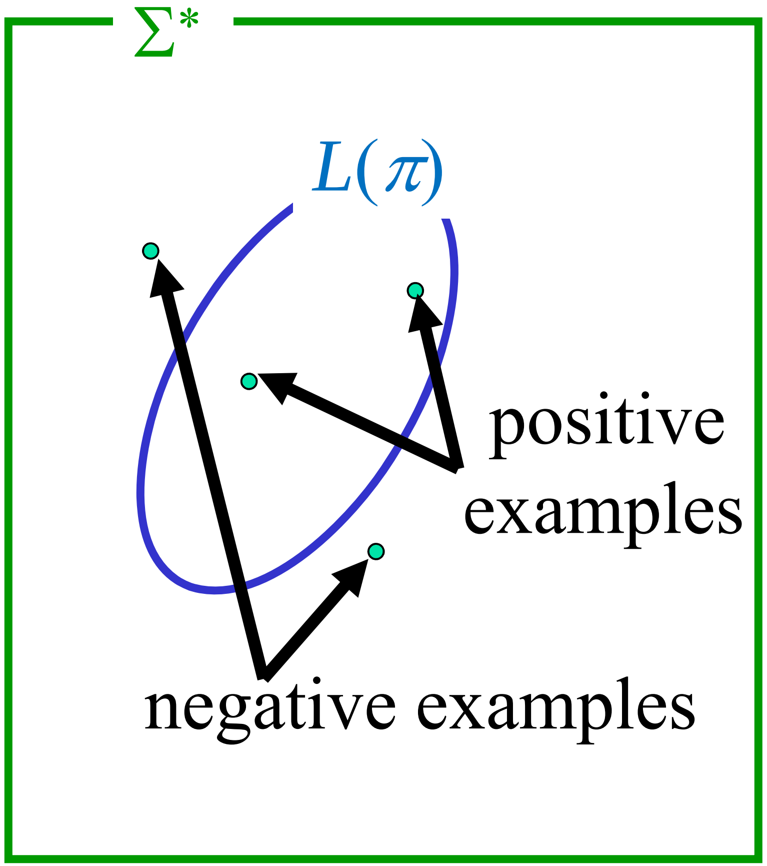
$L(\pi)$: a language represented with a pattern π

- a positive example on $L(\pi)$:

  $< s, +>$ for $x \in L(\pi)$

  a negative example on $L(\pi)$:

  $< s, -\!>$ for $x \notin L(\pi)$



Abstract Classification

- A half-plane P which contains C (yes) and excludes D (no) is to be learned
- The half-plane P is represented as a pair $(\boldsymbol{w}, c)$ which means the linear inequation $(\boldsymbol{w}, \boldsymbol{x}) + c > 0$.
  - Let $C(p)=\{\boldsymbol{x} \in \boldsymbol{R}^n \mid p(\boldsymbol{x})\}$ for a predicate p.

    Then the search space (version space) is

    $$\mathrm{C} = \{C(\lambda \boldsymbol{x}.((\boldsymbol{w}, \boldsymbol{x}) + c > 0)) \mid \boldsymbol{w} \in \boldsymbol{R}^n , c \in \boldsymbol{R}^n \}.$$

    The set of parameter s are from

    $$\mathrm{H} = \{ (\boldsymbol{w}, c) \mid \boldsymbol{w} \in \boldsymbol{R}^n , c \in \boldsymbol{R}^n \}.$$
- The training examples are provided as the sets C and D.
- A learning algorithm is provided.

Typical evaluation method

- A learning algorithm A is evaluated with test data as follows.

Step1. Let C_* are set of all positive data and D_* be are all negatives.

Step 2. Select subsets $C_{\text{training}} \subset C_*$ and $D_{\text{training}} \subset D_*$ for training.

Step 3. Apply A to the pair C_{training} and D_{training} and obtain a rule f.

Step 4. Select subsets C_{test} and D_{test} make a confusion matrix.

Step 5. Calculate some measures from the confusion matrix.

Confusion Matrix

- Every data is represented as a pair **x** = <*s*, *p*>

  $p = +$ if $s \in C$ and $p = -$ if $s \in D$

	C_{test}	D_{test}
$\{s \in C_{\text{test}} \cup D_{\text{test}} \mid f(s) = 1\}$ positive	true positive	false positive
$\{s \in C_{\text{test}} \cup D_{\text{test}} \mid f(s) = 0\}$ negative	false negative	true negative

Measures

- Accuracy $$\frac{|TP|+|TN|}{|TP|+|FP|+|TN|+|FN|}$$
- Error rates 1- Accuracy
- Precision (positive predictive rate)
$$\frac{|TP|}{|TP|+|FP|}$$
- Recall (coverage, true-positive rate, sensitivity) $$\frac{|TP|}{|TP|+|FN|}$$

When learning is correct?

- Every data is represented as a pair **x** = $<s, p>$

  $p = +$ if $s \in C$ and $p = -$ if $s \in D$

	C_{test}	D_{test}
$\{s \in C_{test} \cup D_{test} \mid f(s) = 1\}$ positive		empty
$\{s \in C_{test} \cup D_{test} \mid f(s) = 0\}$ negative	empty	

Comparison with an Unknown Function

- Assuming an unknown discriminant function f_* such that

$$C_* = \{ \boldsymbol{x} = \langle w, 1\rangle \mid f_*(w) = 1 \}$$

$$D_* = \{ \boldsymbol{x} = \langle w, 1\rangle \mid f_*(w) = 0 \}$$

we evaluate the learning algorithm A by comparing its output f with f_* .

- If every function f that we treat is represented as a parameter p, we compare p for f and p_* for f_* .
  - Every linear inequation $(\boldsymbol{w}, \boldsymbol{x}) + c > 0$ is represented as a parameter vector $(\boldsymbol{w}, c)$.
  - We evaluate A with comparing $(\boldsymbol{w}, c)$ and $(\boldsymbol{w}_*, c_*)$.

Correctness with Unknown Functions (1)

- Assuming an unknown discriminant function f_* , we could say that the learning algorithm A is correct if the output f of A becomes *nearer* f_* when more data are fed to A.
- Mathematically, consider a infinite sequence of training data sets $(C_0, D_0), (C_1, D_1), (C_2, D_2), \ldots$ such that

$$C_0 \subset C_1 \subset C_2 \subset \ldots \subset C_* \text{ and}$$

$$D_0 \subset D_1 \subset D_2 \subset \ldots \subset D_* .$$

Let f_i be the output of A for C_i and D_i.

Then the algorithm A is correct if $\| f_i - f_* \| \to 0$ for any of such sequences.

Correctness with Unknown Functions (2)

- A similar definition of correctness could be defined:
  If the learning algorithm A is correct if
  A outputs f_* whenever an enough amount of training data are fed to A.
- Mathematically, consider a infinite sequence of training data sets $(C_1, D_1), (C_2, D_2), (C_3, D_3), \ldots$ such that
  $$C_1 \subset C_2 \subset C_3 \subset \ldots \subset C_* \text{ and}$$
  $$D_1 \subset D_2 \subset D_3 \subset \ldots \subset D_*.$$
  Let f_i be the output of A for C_i and D_i.
  Then the algorithm A is correct if for each of such sequences, there exists an N such that $\| f_i - f_* \| = 0$ for all $n \geq N$.

Estimation and Learning

- Estimation in statistics means to infer the value of parameters from examples.
- We assume an unknown value of θ.
- The parameter θ affects the distribution of $D(\theta)$, and only finite number of data are coming from the set.
- We expect that, more data from $D(\theta)$, better conjecture θ^ could be obtained.
- The conjecture θ ^ is (statistically) consistent if

$$\lim_{n\to\infty} \mathrm{E}(\theta\,\hat{}\,) = \theta$$

Correctness of Learning Patterns

Examples on $L(\pi)$

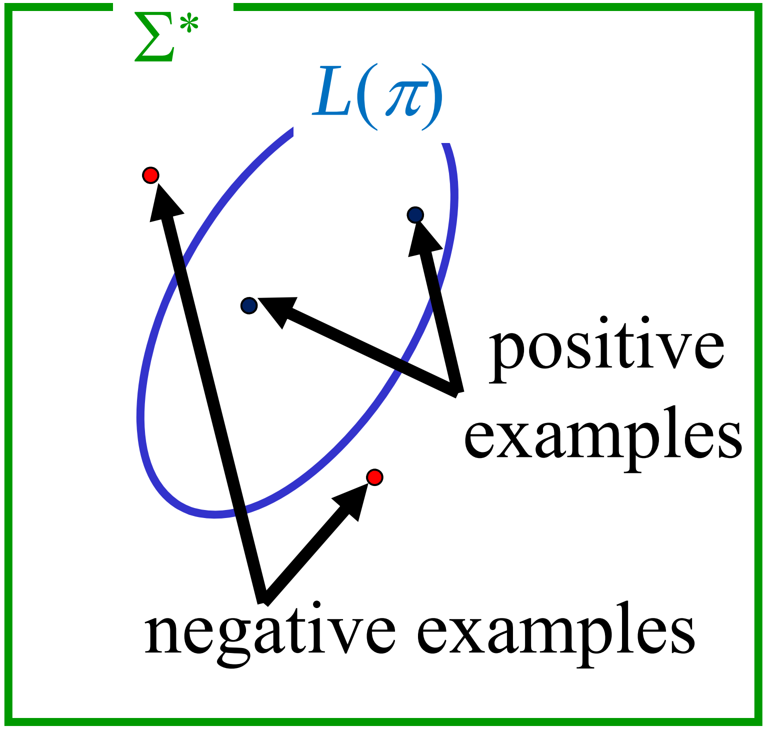
- We assume that, for an unknown pattern π_*,

C_* is a finite set of positive examples on $L(\pi_*)$ and
D_* is a finite set of negative examples on $L(\pi_*)$.

- a positive example on $L(\pi)$:

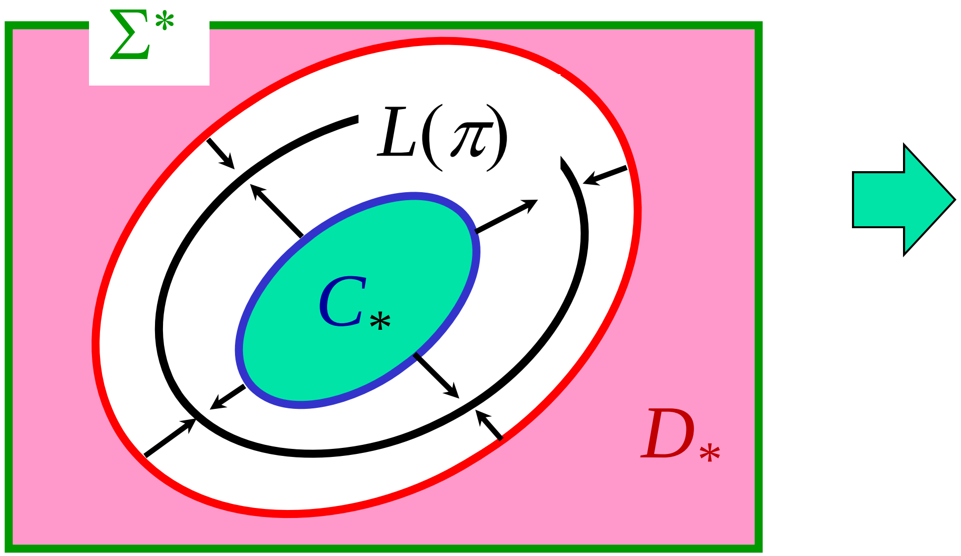
  $< x, +>$ for $x \in L(\pi)$
- a negative example on $L(\pi)$:

  $< x, -> $ for $x \notin L(\pi)$



Question

- If we give more and more (negative and positive) examples on $L(\pi_*)$ to an learning algorithm, does it eventually conjecture the unknown π_*?
- We have to give mathematical definitions of
  - giving more and more examples, and
    - or giving examples many enough
  - conjecturing π eventually.



$\pi^\wedge \to \pi$

Assumption

- Without loss of generality, we may assume that learning algorithm takes examples in C_* and D_* one by one.
- In the situation that both C_i and D_i grow, we assume that an infinite sequence σ of strings marked with either + or –, and some truncation of σ corresponds to C_i and D_i.

Example

σ : <ab,+>, <aab,+>, <bbb,–>, <aaab,+>, <abba,–>, ..

C_i = {ab, aab, aaab},

D_i = {bbb, abba}.

Presentations

Definition A presentation of $L(\pi)$ is a infinite sequence

σ: $< s_0, p_0 >, < s_1, p_1 >, < s_2, p_2 >, \ldots$

where $s_i \in \Sigma^*$ and $p_i = +$ or $-$.

- $< s, + >$ is a positive example
- $< s, - >$ is a negative example

- $\sigma[n] = < s_0, p_0 >, < s_1, p_1 >, < s_2, p_2 >, \ldots, < s_{n-1}, p_{n-1} >$

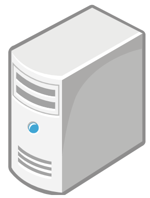
Definition A presentation σ is complete if

any $x \in L(\pi)$ appears in σ as a positive example $< x, + >$ at least once and

any $x \notin L(\pi)$ appears in σ as a negative example $< x, - >$ at least once.

Identification in the limit [Gold]

$x_1, x_2, x_3, \ldots$ ➡



$\pi_1, \pi_2, \pi_3, \ldots$

- A learning algorithm A EX-identifies $L(\pi)$ in the limit from complete presentations if
  for any complete presentation $\sigma = x_1, x_2, x_3, \ldots$ of $L(\pi)$
  and the output sequence $\pi_1, \pi_2, \pi_3, \ldots$ of A, there exists N
  such that for all $n \geq N$ $\pi_n = \pi'$ and $L(\pi') = L(\pi)$

The learning algorithm *learn-patterns*

For $n = 1$ forever

receive $e_n = \langle s_n , b_n \rangle$

compute the list $l = \pi_1, \pi_2 , \ldots, \pi_k$ of all the least common anti-unifications of the set of positives $C_n = \{s_j : \langle s_j , + \rangle$ and $j = 1, 2, \ldots, n\}$

for each π_j in the list l

if an $e_j = \langle s_j , - \rangle$ s.t. $s_j \in L(\pi_j)$ is found

delete π_j from l

if l is not empty

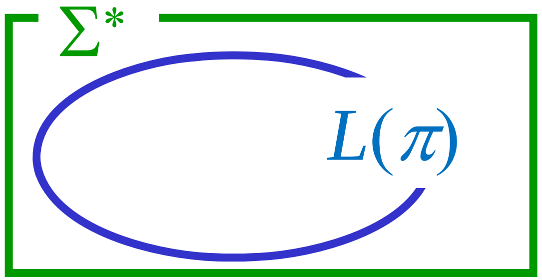
return one π_i

Identification of patterns

Theorem The algorithm of *Learn-pattern* EX-identifies the class of all pattern languages in the

Positive presentations

$e_1, e_2, e_3, \ldots$



$\pi_1, \pi_2, \pi_3, \ldots$

- A presentation σ is positive if σ consists only of positive example $< s, +>$ and any positive example occurs at least once in σ.
- A presentation σ is complete if any $x \in L(\pi)$ appears in σ as a positive example $< s, +>$ at least once.

The learning algorithm *learn-patterns*

For $n = 1$ forever

receive $e_n = \langle s_n, b_n \rangle$

compute the list $l = \pi_1, \pi_2, \ldots, \pi_k$ of all the least common anti-unifications of the set of positives $C_n = \{s_j : \langle s_j, + \rangle \text{ and } j = 1, 2, \ldots, n\}$

~~for each π_j in the list l~~

~~if an $e_j = \langle s_j, - \rangle$ s.t. $s_j \in L(\pi_j)$ is found~~

~~delete π_j from l~~

~~if l is not empty~~

return one π_i

Identification of patterns

Theorem The revised algorithm of *Learn-pattern* EX-identifies the class of all pattern languages in the limit from positive presentations.