Logic and Knowledge Representation

Reinforcement learning, Inductive Logic Programming

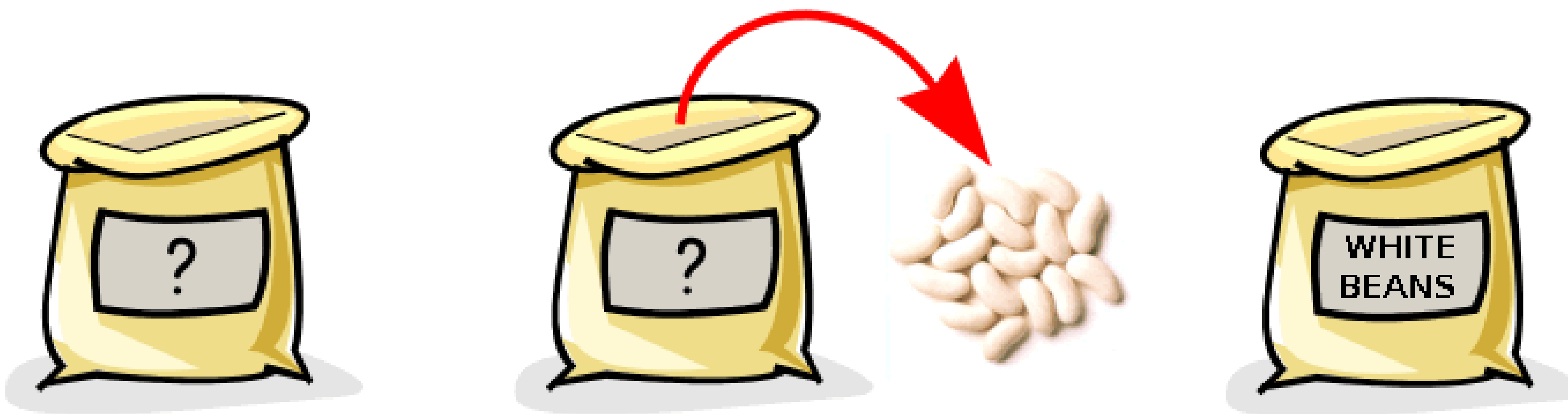
Description Complexity

15 June 2018

Giovanni Sileno gsileno@enst.fr

Télécom ParisTech, Paris-Dauphine University

Induction (again) – after Pierce



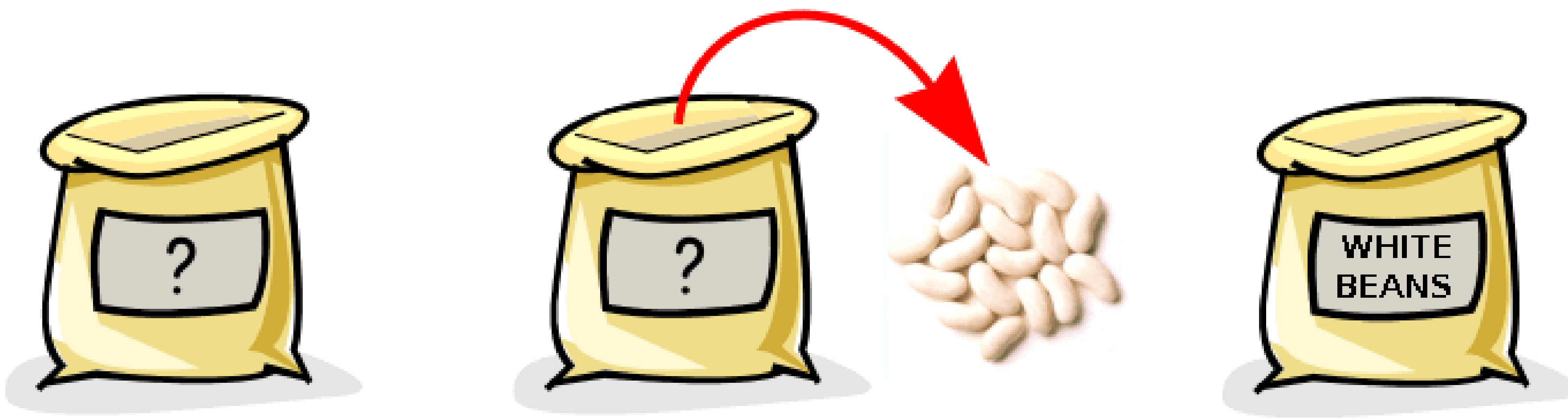
Induction

Fact: These beans are from this bag.

Fact: These beans are white.

⇒ Hyp. rule: All the beans from this bag are white.

Induction (again) – after Pierce



Induction

Fact: These beans are from this bag.

Fact: These beans are white.

⇒ Hyp. rule: All the beans from this bag are white.

- Induction enables ***prediction*** through the settled model.

Induction (again)

3, 4, 6, 8, 12, 14, 18, 20, 24, 30, 32, 38, 42, ... **??**

Possible models?

Induction (again)

3, 4, 6, 8, 12, 14, 18, 20, 24, 30, 32, 38, 42, ... **??**

Possible models:

- numbers n + 1, n prime number $\longrightarrow$ **44, 48, 54, ..**

Induction (again)

3, 4, 6, 8, 12, 14, 18, 20, 24, 30, 32, 38, 42, ... **??**

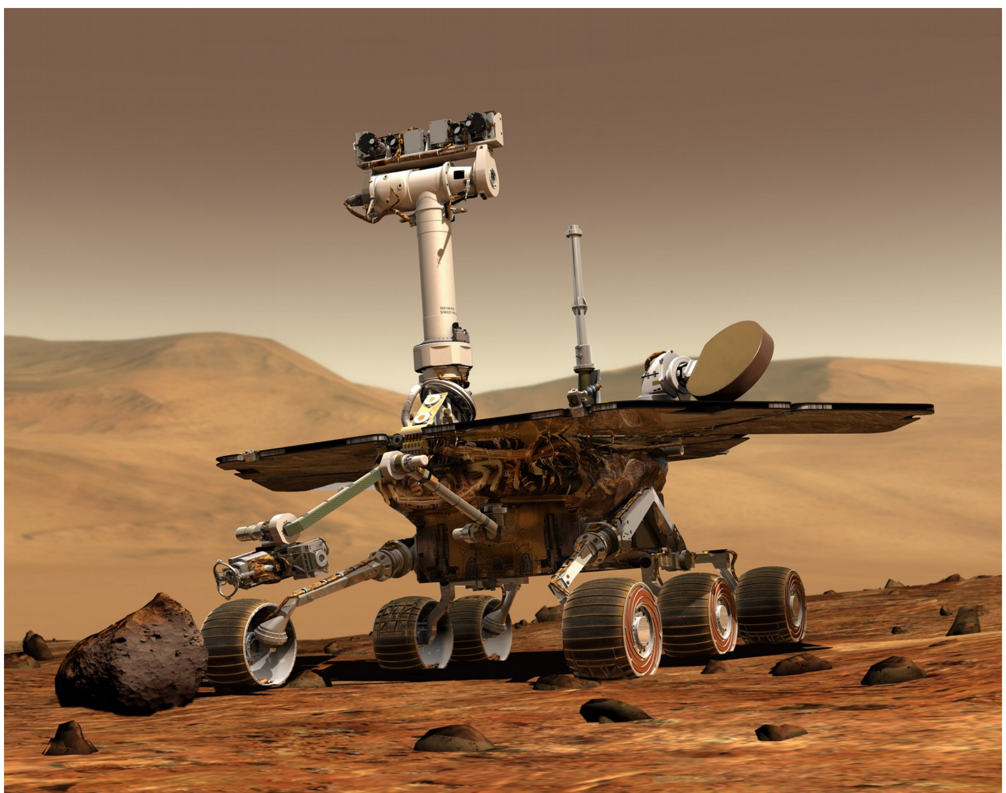
Possible models:

- numbers n + 1, n prime number $\rightarrow$ **44, 48, 54, ..**
- numbers n such that for all k with gcd(n, k) = 1 and $n > k^2$, $n - k^2$ is prime. $\rightarrow$ **48, 54, 60, ..**

- Further observations enable the correction of the model.

Alien environment problem

- Suppose a robot lands on an unknown planet.
  - in order to accomplish its mission, it has to acquire an operational knowledge of:



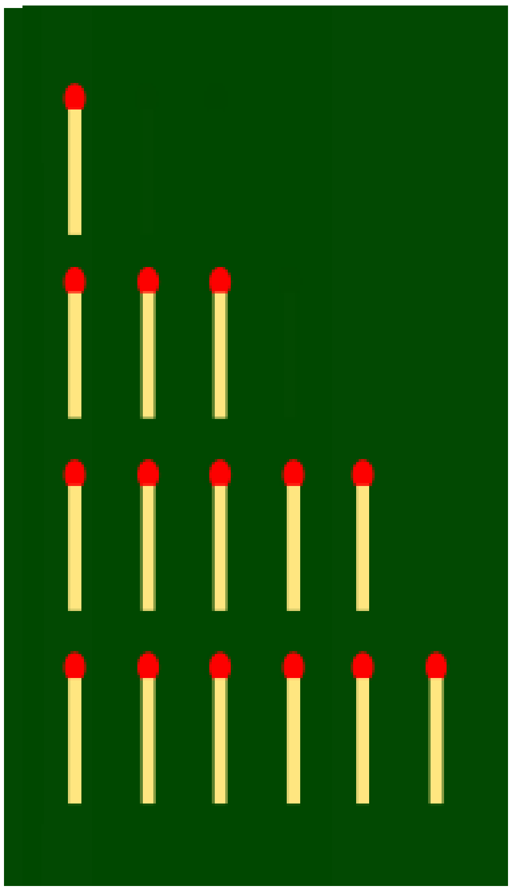
- what (might) occur
- what its actions (might) achieve

*from its **observations**!!!*

Induction

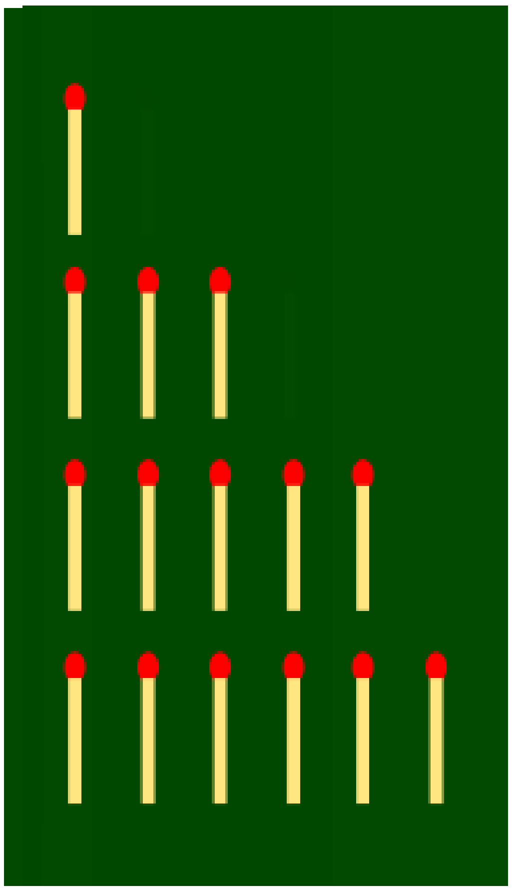
Reinforcement Learning

Nim game



- Two players game
- Each player may take as many items from a single row in turn
- The one who takes the last item loses.

Nim game



- How one can learn to win without knowing the rules?
  - recording states encountered during the play
  - updating value of states with final results (won or lost)
  - selecting actions bringing to winning states

very simple example of **reinforcement learning**!

Example of reinforcement learning algorithm

state of the world

expected immediate reinforcement
in performing a in s

expected (long-term) gain
from a state s'

$$\pi^*(s) = \mathrm{argmax}_a\,(R(s,a) + \gamma V^*(\delta(s,a)))$$

best strategy in s

action

discount factor

transition function:
a performed in s
leads to s'

Example of reinforcement learning algorithm

state of the world

expected immediate reinforcement
in performing a in s

expected (long-term) gain
from a state s'

$$\pi^*(s) = \operatorname{argmax}_a \left(R(s,a) + \gamma V^*(\delta(s,a))\right)$$

best strategy in s

action

discount factor

transition function:
a performed in s
leads to s'

$$Q(s, a) = R(s,a) + \gamma V^*(\delta(s,a))$$

Utility function

expected gain

Q-learning

$$Q(s, a) = R(s,a) + \gamma\ V^*(\delta(s,a))$$

$$= R(s,a) + \gamma\ \max_{a'}(Q(s', a'))$$

$$= R(s,a) + \gamma\ \max_{a'}(Q(\delta(s,a), a'))$$

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

Q-learning algorithm

$Q(s, a) = R(s,a) + \gamma \max_{a'}(Q(\delta(s,a), a'))$

$\pi^*(s) = \text{argmax}_a Q(s, a)$

```
initialize the table Q(s,a) to zero
   observe the current state s.
   repeat
      choose an action and execute it
      receive the reward r
      observe the new state s'
      update the table Q(s,a) as:
         Q(s,a) := r + max_a' Q(s',a')
         s := s'
```

This was about behaviour,
but what about knowledge?

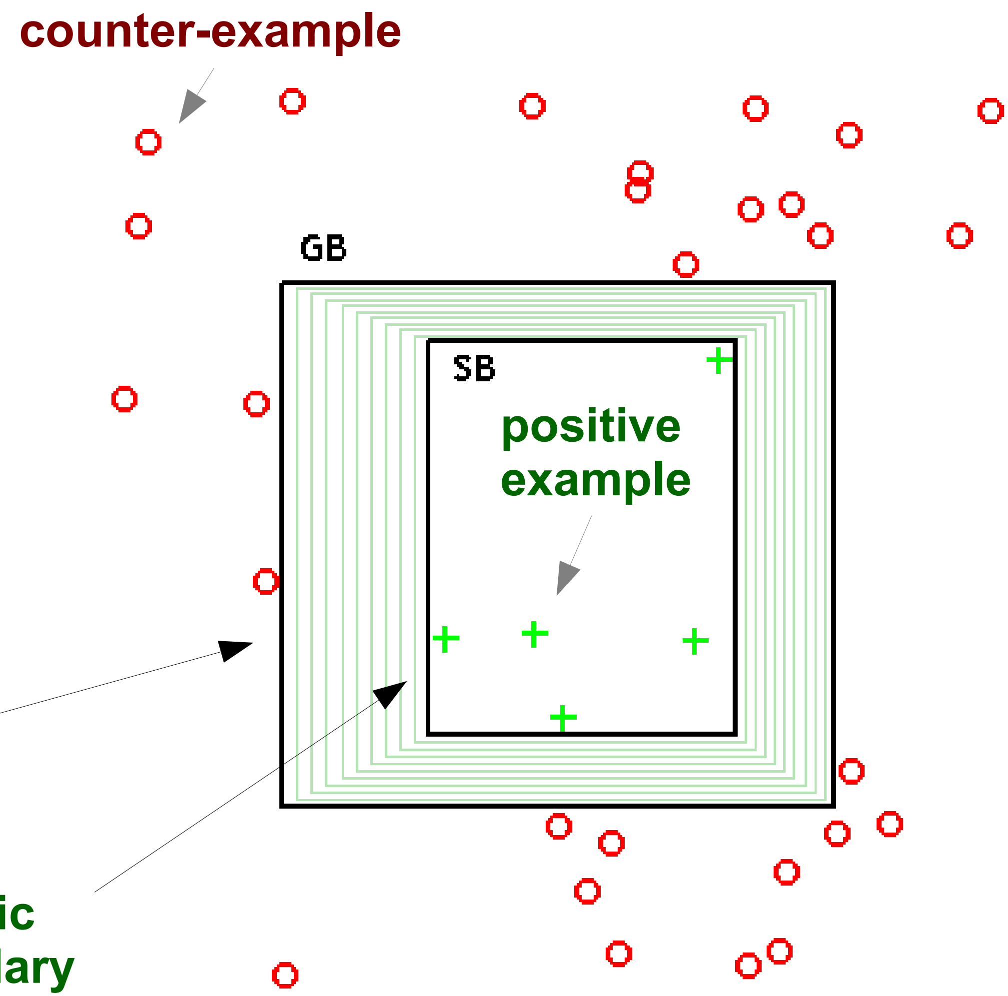
Induction as ***generalization...***

Version space learning

- Logical approach to *binary classification*
- Search on a predefined space of hypotheses:

  $$H_1 \vee H_2 \vee \ldots \vee H_n$$

- You do not need to maintain exemplars!



[Dubois, Vincent; Quafafou, Mohamed (2002). "Concept learning with approximation: Rough version spaces". RSCTC 2002. Sverdlik, W.; Reynolds, R.G. (1992). "Dynamic version spaces in machine learning". TAI '92.]

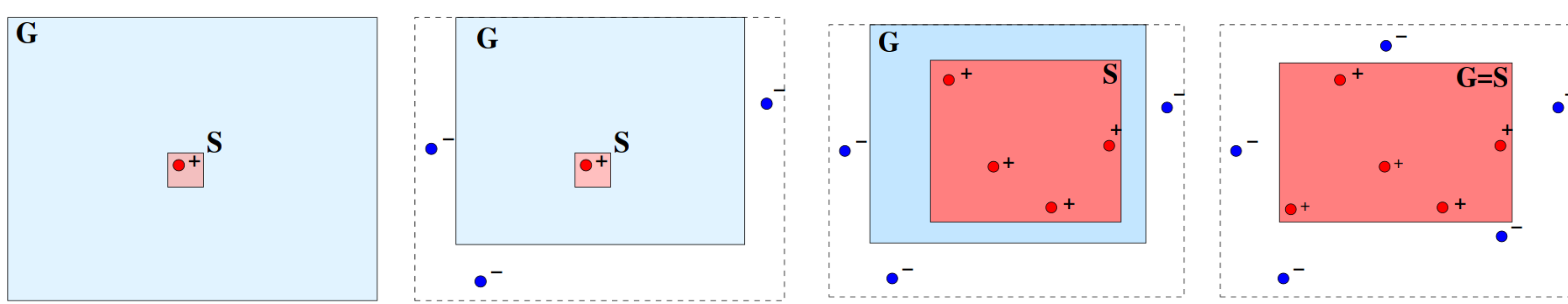
Using a version space

current example

current hypothesis, taken from a version space

```
represent E
predict from the representation of H
whether or not E exemplifies H
   if correct then retain H
   if incorrect then
      identify the differences between E and H
      use the selected differences to
       - generalize H if it is a positive instance
       - specialize H if it is a negative instance
```

candidate elimination algorithm



Machine learning

Machine learning

Machine learning is a process that enables artificial systems to improve with experience.

what are the criteria?

Machine learning

Machine learning is a process that enables artificial systems to improve with experience.

- Elements of a learning task
  - Items of Experience, $i \in I$
  - Available Actions: $a \in A$
  - Evaluation: $v(a, I)$
  - Performer System: $b: I \rightarrow A$
  - Learning System: $L: (i_1, a_1, v_1)\ldots(i_n, a_n, v_n) \rightarrow b$

Types of learning problems

- batch or offline vs online learning

  training phase and testing *vs* learning while doing

- complete vs partial vs pointwise feedback

  feedback concerns all vs some vs one performer system

- passive vs active learning

  observation vs experimentation

- acausal or casual setting

  presence or not of side-effects: e.g. rain prediction vs behavioural control

- stationary vs non-stationary environment

  evaluation does or does not change in time

Learning a function from examples

domain X: descriptions

domain Y: predictions

H: hypothesis space

h: target hypothesis

examples

(x_1, y_1)

(x_2, y_2)

..

(x_n, y_n)

→ learner →

function

$h \in H$, $h: X \to Y$

Learning a function from examples

domain X: descriptions

domain Y: predictions

H: hypothesis space

h: target hypothesis

examples

(x_1, y_1)

(x_2, y_2)

..

(x_n, y_n)

learner

function

$h \in H$, $h: X \rightarrow Y$

- Many learning methods are available, but studied and used by different communities!
- A few examples...

Learning a function from examples

domain X: descriptions

domain Y: predictions

H: hypothesis space

h: target hypothesis

examples

(x_1, y_1)

(x_2, y_2)

..

(x_n, y_n)

learner

function

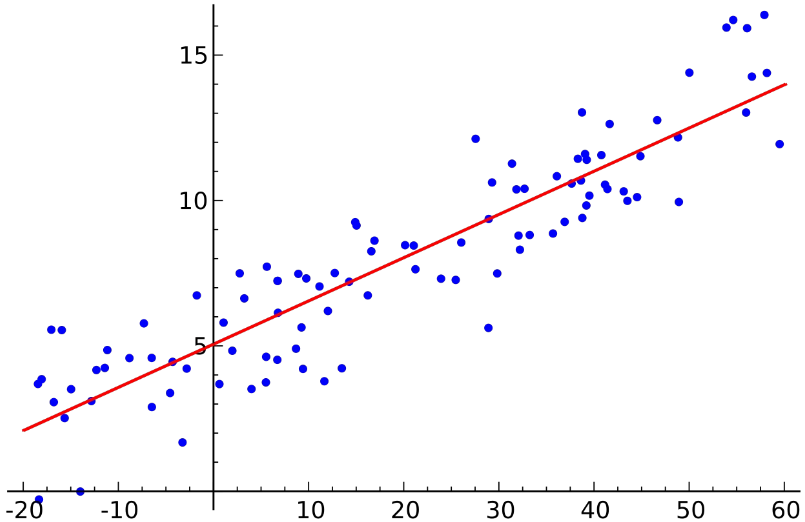
$h \in H$, $h: X \rightarrow Y$

- Method 1: traditional statistics (regression analysis)

  $h: R^n \longrightarrow R$

  h is a linear function

  squared prediction error



Learning a function from examples

domain X: descriptions

domain Y: predictions

H: hypothesis space

h: target hypothesis

examples

(x_1, y_1)

(x_2, y_2)

..

(x_n, y_n)

learner

function

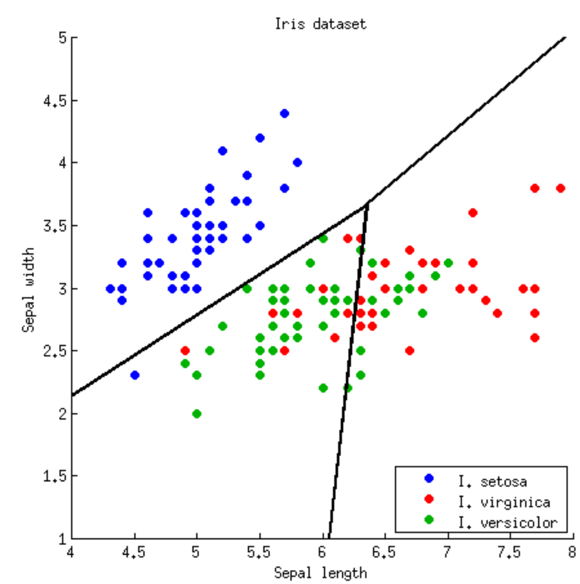
$h \in H$, $h: X \rightarrow Y$

- Method 2: traditional pattern recognition

  $h: R^n \rightarrow \{0, 1, \ldots, m\}$

  h is a discriminant boundary

  right/wrong prediction error



Learning a function from examples

domain X: descriptions

domain Y: predictions

H: hypothesis space

h: target hypothesis

examples

(x_1, y_1)

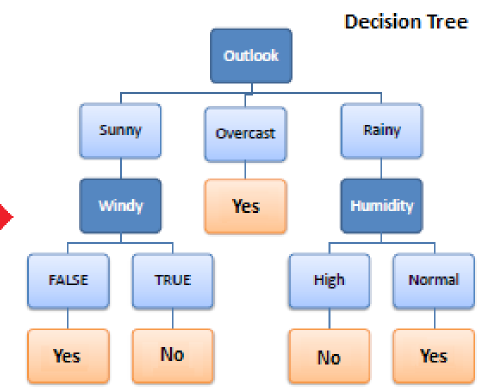
(x_2, y_2)

..

(x_n, y_n)

→ learner → *function* $h \in H,\ h: X \rightarrow Y$

Predictors				Target
Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



- Method 3: "symbolic" machine learning

  h: {attribute-value vectors} $\rightarrow$ {0, 1}

  h is a boolean function (e.g. a decision tree)

Learning a function from examples

domain X: descriptions

domain Y: predictions

H: hypothesis space

h: target hypothesis

examples

(x_1, y_1)

(x_2, y_2)

..

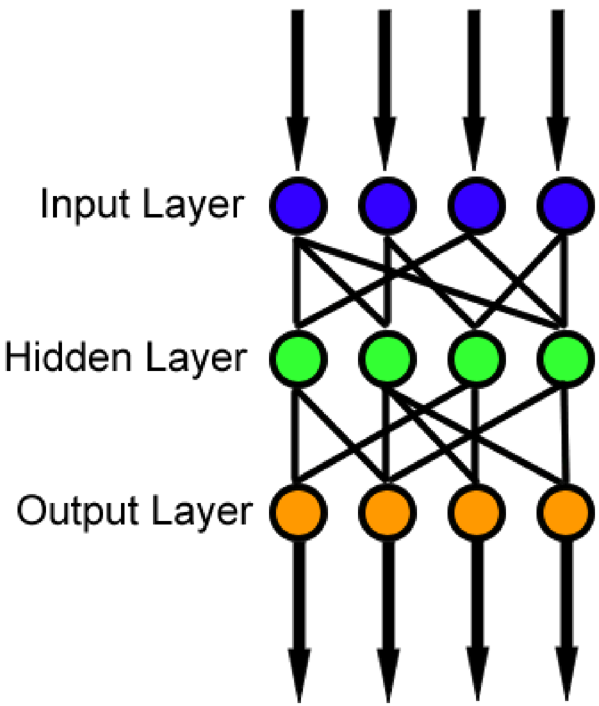
(x_n, y_n)

→ learner →

function

$h \in H,\ h: X \rightarrow Y$

- Method 4: Neural networks
  - $h: R^n \rightarrow R$
  - h is a feedforward neural net



Learning a function from examples

domain X: descriptions

domain Y: predictions

H: hypothesis space

h: target hypothesis

examples

(x_1, y_1)

(x_2, y_2)

..

(x_n, y_n)

learner

function

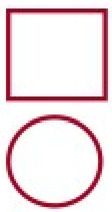
$h \in H$, $h: X \rightarrow Y$

- Method 5: Inductive Logic Programming
  - h: {term structure} $\longrightarrow$ {0, 1}
  - h is a "simple" logic program.

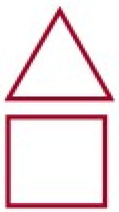
Inductive Logic Programming

Symbolic induction

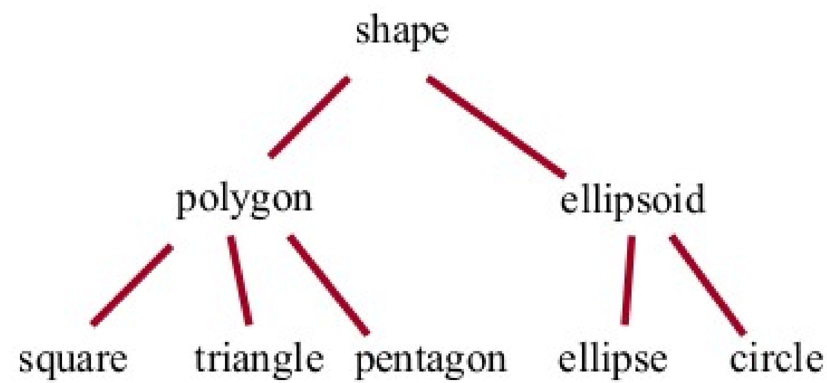
Exemplar of class X



Exemplar of class X



Background knowledge




```
E1 = square(A) & circle(B) & above(A, B)
E2 = triangle(C) & square(D) & above(C, D)
```

What is X?

a group of geometric shapes?

a group of 2 geometric shapes?

a group of 2 geometric shapes with a square?

- Induction as *least general generalization* of exemplars.

Inductive Logic Programming

- Examples:

```
cute(X) :- dog(X), small(X), fluffy(X).
cute(X) :- cat(X), fluffy(X).
```

- Generalisation:

```
cute(X) :- fluffy(X).
```

Inductive Logic Programming

- Examples:

```
cute(X) :- dog(X), small(X), fluffy(X).
cute(X) :- cat(X), fluffy(X).
```

- Generalisation:

```
cute(X) :- fluffy(X).
```

- Background knowledge:

```
pet(X) :- cat(X).
pet(X) :- dog(X).
small(X) :- cat(X).
```

- Generalisation:

```
cute(X) :- pet(X), small(X), fluffy(X).
```

Inductive Logic Programming

- Examples E are expected to result from background knowledge B and hypothesis H:

  $B \land H \models E$

- **Inverse resolution**:
  - from example:

    ```
    cute(X) :- cat(X), fluffy(X).
    ```

  - from knowledge:

    ```
    pet(X) :- cat(X).
    small(X) :- cat(X).
    ```

  induce:

  ```
  cute(X) :- pet(X), small(X), fluffy(X).
  ```

Explanation-Based Generalization

Explanation-based Generalization

```
telephone(T) :- connected(T), partOf(T, D),
   dialingDevice(D), emitsSound(T).

connected(X) :- hasWire(X, W), attached(W, wall).
connected(X) :- feature(X, bluetooth).
connected(X) :- feature(X, wifi).
connected(X) :- partOf(X, A), antenna(A),
   hasProtocol(X, gsm).

dialingDevice(DD) :- rotaryDial(DD).
dialingDevice(DD) :- frequencyDial(DD).
dialingDevice(DD) :- touchScreen(DD),
   hasSoftware(DD,DS), dialingSoftware(DS).

emitsSound(P) :- hasHP(P).
emitsSound(P) :- feature(P, bluetooth).
```

Explanation-based Generalization

```
example(myphone, Features) :-
    Features = [silver(myphone),
                belongs(myphone, jld),
                partOf(myphone, tc), touchScreen(tc),
                partOf(myphone, a), antenna(a),
                hasSoftware(tc, s1), game(s1),
                hasSoftware(tc, s2),
                dialingSoftware(s2),
                feature(myphone,wifi),
                feature(myphone,bluetooth),
                hasProtocol(myphone, gsm),
                beautiful(myphone)].
```

- Features activated during the proof:

```
[ feature(myphone, bluetooth), partOf(myphone, tc),
  touchScreen(tc), hasSoftware(tc, s2),
  dialingSoftware(s2), feature(myphone, bluetooth) ]
```

Explanation-based Generalization

- Features activated during the proof:

```
[ feature(myphone, bluetooth), partOf(myphone, tc),
  touchScreen(tc), hasSoftware(tc, s2),
  dialingSoftware(s2), feature(myphone, bluetooth) ]
```

Explanation-based Generalization

- Features activated during the proof:

```
[ feature(myphone, bluetooth), partOf(myphone, tc),
  touchScreen(tc), hasSoftware(tc, s2),
  dialingSoftware(s2), feature(myphone, bluetooth) ]
```

- From the trace, by generalizing shared constants:

```
C001(X) :- feature(X, bluetooth), partOf(X, Y),
   touchScreen(Y), hasSoftware(Y, Z), dialingSoftware(Z).
```

Explanation-based Generalization

- Features activated during the proof:

```
[ feature(myphone, bluetooth), partOf(myphone, tc),
  touchScreen(tc), hasSoftware(tc, s2),
  dialingSoftware(s2), feature(myphone, bluetooth) ]
```

- From the trace, by generalizing shared constants:

```
C001(X) :- feature(X, bluetooth), partOf(X, Y),
   touchScreen(Y), hasSoftware(Y, Z), dialingSoftware(Z).
```

- By grouping predicates that do not depend on X:

```
C002(Y) :- touchScreen(Y), hasSoftware(Y, Z),
   dialingSoftware(Z).
```

Explanation-based Generalization

- Features activated during the proof:

```
[ feature(myphone, bluetooth), partOf(myphone, tc),
  touchScreen(tc), hasSoftware(tc, s2),
  dialingSoftware(s2), feature(myphone, bluetooth) ]
```

- From the trace, by generalizing shared constants:

```
C001(X) :- feature(X, bluetooth), partOf(X, Y),
   touchScreen(Y), hasSoftware(Y, Z), dialingSoftware(Z).
```

- By grouping predicates that do not depend on X:

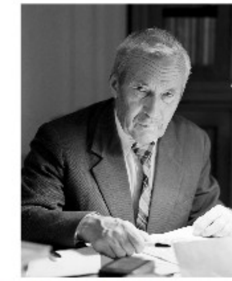
```
C002(Y) :- touchScreen(Y), hasSoftware(Y, Z),
   dialingSoftware(Z).
```

- C001 then becomes

```
C001(X) :- feature(X, bluetooth), partOf(X, Y), C002(Y).
```

Description Complexity

Informal definition of Kolmogorov complexity

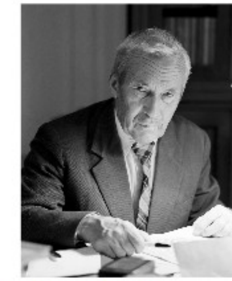


Andrei Kolmogorov

- The complexity of an object corresponds to the **minimal length of a computer program** producing this object.

[informal reduction of a presentation by Pierre-Alexandre Murena (Télécom ParisTech)]

Informal definition of Kolmogorov complexity



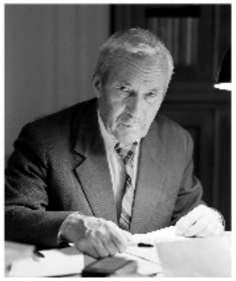
Andrei Kolmogorov

- The complexity of an object corresponds to the **minimal length of a computer program** producing this object.

- A finite string like “aaa…” is not very complex:

```
for i=1..n:
   print a
```

Informal definition of Kolmogorov complexity



Andrei Kolmogorov

- The complexity of an object corresponds to the **minimal length of a computer program** producing this object.

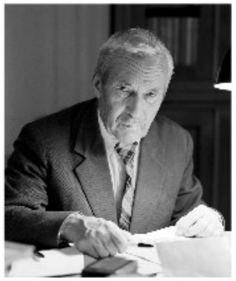
- A finite string like "aaa…" is not very complex:

```
for i=1..n:
   print a
```

- Is π complex?

$$\pi/4 = 1 - 1/3 + 1/5 - 1/7 + 1/9 - \ldots$$

Informal definition of Kolmogorov complexity



Andrei Kolmogorov

- The complexity of an object corresponds to the **minimal length of a computer program** producing this object.

- A finite string like "aaa…" is not very complex:

```
for i=1..n:
   print a
```

- Is π complex?

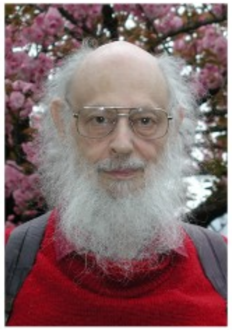
$$\pi/4 = 1 - 1/3 + 1/5 - 1/7 + 1/9 - \ldots$$

Kolmogorov complexity is incomputable.

Randomness


TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
NINE NINE NINE NINE NINE NINE
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS: YOU CAN NEVER BE SURE.
10/25/01 © 2001 United Feature Syndicate, Inc.

Randomness

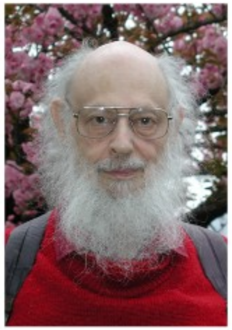


Ray Solomonoff

- Are both these sequences equally random?

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 1 0 0 1 1 0 1 1 0 0 0 1 1 1 0 1 0 1 1 0 0 1 0

Randomness and compression



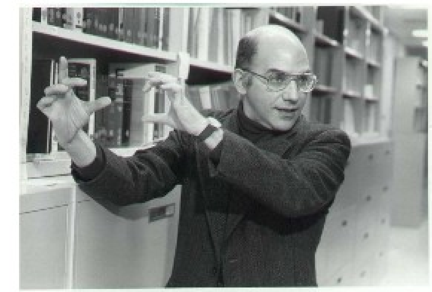
Ray Solomonoff

- Are both these sequences equally random?

    0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
    1 0 0 1 0 0 1 1 0 1 1 0 0 0 1 1 1 0 1 0 1 1 0 0 1 0

- A finite sequence is said to be **random** if it is ***incompressible***, i.e. if its *shortest description* is the sequence itself.

Deduction


Gregory Chaitin

- Deduction generally works
  from the general to the particular

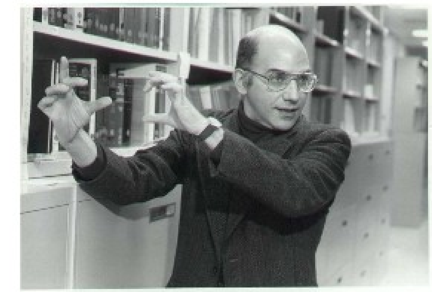
  general premise
  and particular premise → particular conclusion

```
all animals eat
fido is an animal → fido eats.
```

  general premise
  and less general premises → less general conclusion

```
all animals eat
cats are animals → cats eat.
```

Deduction and compression

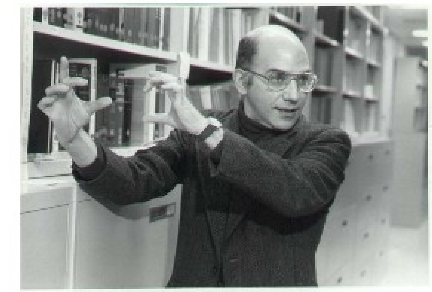


Gregory Chaitin

- Deduction generally works
  from the general to the particular

- Intuition: A formal system is a ***compression*** of the set of theorems it can prove.

Deduction and compression



Gregory Chaitin

- Deduction generally works
  from the general to the particular

- Intuition: A formal system is a ***compression*** of the set of theorems it can prove.

Understanding is compressing.

Minimum Description Length as *inductive principle*

- The MDL principle states that:
  the ***best theory*** to describe observed data is the one which minimizes the sum of the description length (in bits) of:
  - the theory description
  - the data encoded from the theory

Hofstadter’s problems

problems of ***analogy***

ABC : ABD :: IJK : x

RST : RSU :: RRSSTT : x

ABC : ABD :: BCA : x

ABC : ABD :: AABABC : x

IJK : IJL :: IJJKKK : x

Hofstadter's problems

problems of ***analogy***

ABC : ABD :: IJK : x

RST : RSU :: RRSSTT : x

ABC : ABD :: BCA : x

ABC : ABD :: AABABC : x

IJK : IJL :: IJJKKK : x

```
// ABC : ABD :: IJK : IJL
let(alphabet, shift, ?, sequence, 3),
 let(mem,, ?, next_block, mem,, ?, last, increment),
 mem,,, next_block, mem,, 8;

// ABC : ABD :: IJK : IJD
let(alphabet, shift, ?, sequence, 3),
 let(mem,, ?, next_block, mem,, ?, last, 'd'),
 mem,,, next_block, mem,, 8;
```

- Let us apply the MDL principle to decide x:
  - we need to settle a description language with a set of operators manipulating strings.
  - we interpret the data through the description language
  - we compute the complexity of the hypothetical organizations

Hofstadter's problems

problems of ***analogy***

ABC : ABD :: IJK : x

RST : RSU :: RRSSTT : x

ABC : ABD :: BCA : x

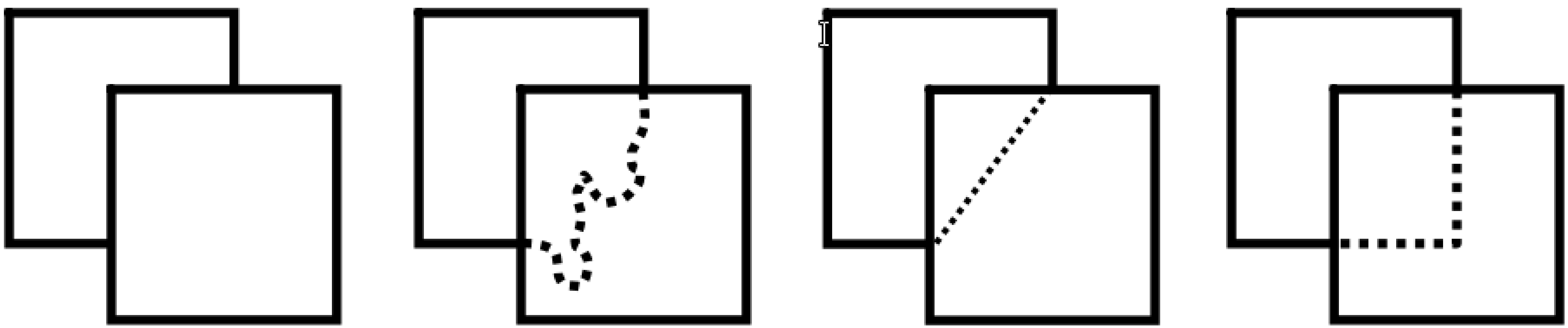
ABC : ABD :: AABABC : x

IJK : IJL :: IJJKKK : x

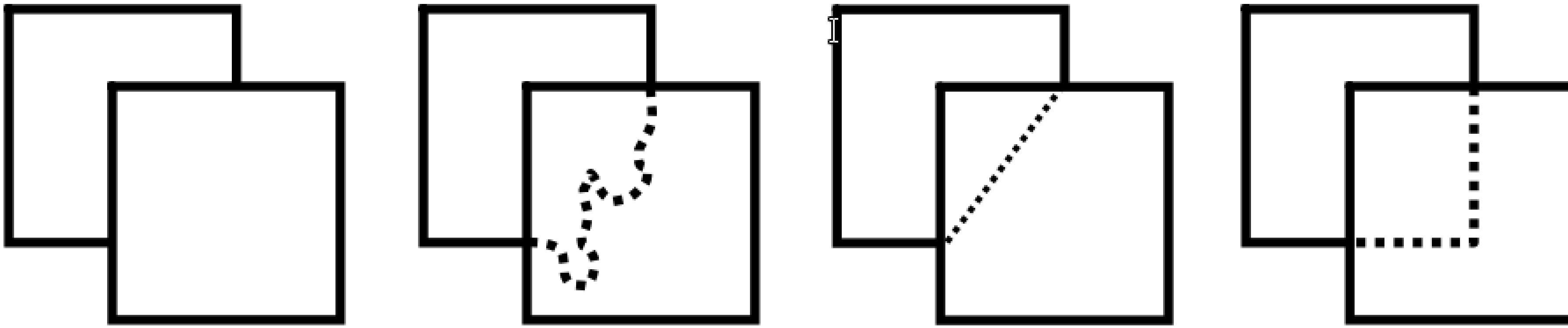
Problem	Solution	Proportion	Complexity
IJK	IJL	93%	37
16.0 ± 0.085 s	IJD	2.9%	38
BCA	BCB	49%	42
21.7 ± 0.12 s	BDA	43%	46
AABABC	AABABD	74%	33
23.8 ± 0.12 s	AACABD	12%	46
IJKLM	IJKLN	62%	40
24.7 ± 0.22 s	IJLLM	15%	41
123	124	96%	27
6.39 ± 0.074 s	123	3%	31
KJI	KJJ	37%	43
18.6 ± 0.13 s	LJI	32%	46
135	136	63%	35
9.93 ± 0.10 s	137	8.9%	37
BCD	BCE	81%	35
21.9 ± 0.30 s	BDE	5.9%	44

Problem	Solution	Proportion	Complexity
IJJKKK	IJJLLL	40%	52
13.7 ± 0.11 s	IJJKKL	25%	53
XYZ	XYA	85%	40
11.2 ± 0.093 s	XYZ	4.4%	34
122333	122444	40%	56
10.0 ± 0.098 s	122334	31%	49
RSSTTT	RSSUUU	41%	54
10.4 ± 0.072 s	RSSTTU	31%	55
IJJKKK	IJJLLL	41%	52
8.67 ± 0.071 s	IJJKKL	28%	53
AABABC	AABABD	72%	33
12.2 ± 0.12 s	AACABD	12%	46
MRRJJJ	MRRJJK	28%	64
22.1 ± 0.18 s	MRRKKK	19%	65
147	148	69%	36
13.6 ± 0.20 s	1410	10%	38

Similar problem...



Similar problem...



..and many many others.