Artificial Intelligence 2

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

Knowledge in learning

So far we learnt a function input → output.

We only assumed to know the form of the function (such as a decision tree) defined by the hypothesis space.

Can we take advantage of **prior knowledge** about the world?

In most cases the prior knowledge is represented as general first-order logical theories.

Some methods:

- current-best-hypothesis search
- version space learning
- inductive logic programming

Logical formulation of learning

Hypotheses, example descriptions, and classification will be represented using **logical sentences**.

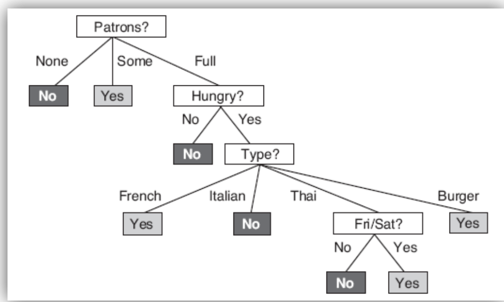
Examples

- **attributes** become unary predicates
  $Alternate(X_1) \land \neg Bar(X_1) \land \neg Fri/Sat(X_1) \land Hungry(X_1) \land \ldots$
- **classification** is given by literal using the **goal predicate**
  $WillWait(X_1)$ or $\neg WillWait(X_1)$

Hypothesis will have the form

$\forall x\ Goal(x) \Leftrightarrow C_j(x)$

C_j is called the extension of the predicate



$\forall r\ WillWait(r) \Leftrightarrow Patrons(r,Some)$
$\lor (Patrons(r,Full) \land Hungry(r) \land Type(r,French))$
$\lor (Patrons(r,Full) \land Hungry(r) \land Type(r,Thai) \land Fri/Sat(r))$
$\lor (Patrons(r,Full) \land Hungry(r) \land Type(r,Burger))$

Hypothesis space

Hypothesis space is the set of all hypothesis.

The learning algorithm believes that one hypothesis is correct, that is, it believes the sentence:

$h_1 \lor h_2 \lor h_3 \lor \ldots \lor h_n$

Hypotheses that are not consistent with the examples can be rules out.

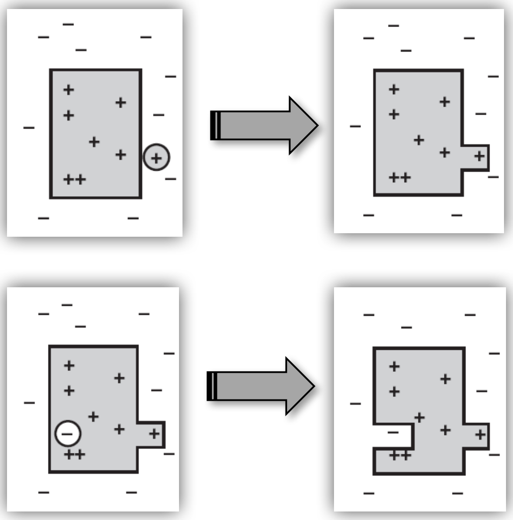
There are two possible ways to be **inconsistent** with an example (the notions originated in medicine to describe erroneous results from lab tests)

- **false negative** – hypothesis says the example should be negative but in fact it is positive
- **false positive** – hypothesis says the example should be positive but in fact it is negative

Current-best-hypothesis search

The idea is to maintain a single hypothesis, and to adjust it as new examples arrive in order to maintain consistency

- if the example is **consistent** with the hypothesis
  - then do **not change** it
- if **false negative**
  - then **generalize** the hypothesis
- if **false positive**
  - then **specialize** the hypothesis



The current-best-hypothesis learning algorithm

```
function CURRENT-BEST-LEARNING(examples, h) returns a hypothesis or fail
  if examples is empty then
    return h
  e ← FIRST(examples)
  if e is consistent with h then
    return CURRENT-BEST-LEARNING(REST(examples), h)
  else if e is a false positive for h then
    for each h′ in specializations of h consistent with examples seen so far do
      h″ ← CURRENT-BEST-LEARNING(REST(examples), h′)
      if h″ ≠ fail then return h″
  else if e is a false negative for h then
    for each h′ in generalizations of h consistent with examples seen so far do
      h″ ← CURRENT-BEST-LEARNING(REST(examples), h′)
      if h″ ≠ fail then return h″
  return fail
```

How to implement specialization and generalization of the hypothesis?

- If hypothesis h_1 is a **generalization** of hypothesis h_2, then we must have
  $\forall x\, C_2(x) \Rightarrow C_1(x)$
- C_i is typically a conjunction of predicates
  - generalization can be realized by **dropping conditions** or by **adding disjuncts**
  - specialization can be realized by **adding extra conditions** or by **removing disjuncts**

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

A restaurant example:

- the first example is positive, attribute Alternate(X_1) is true, so let the initial hypothesis be
  h_1: ∀x WillWait(x) ⇔ Alternate(x)
- the second example is negative, hypothesis predicts it to be positive, so it is a false positive; we need to specialize by adding extra condition
  h_2: ∀x WillWait(x) ⇔ Alternate(x) ∧ Patrons(x,Some)
- the thirst example is positive, the hypothesis predicts it to be negative, so it is a false negative; we need to generalize by dropping the condition Alternate
  h_3: ∀x WillWait(x) ⇔ Patrons(x,Some)
- The fourth example is positive, the hypothesis predicts it to be negative, so it is a false positive; we need to generalize by adding a disjunct (we cannot drop the Patrons condition)
  h_3: ∀x WillWait(x) ⇔ Patrons(x,Some) ∨ (Patrons(x,Full) ∧ Fri/Sat(x))

After each modification of the hypothesis we need to **check all the previous examples**.

There are several possible generalizations and specializations and we may need to **backtrack** where no simple modification of the hypothesis is consistent with all the data.

The source of problems – **strong commitment**

- The algorithm has to choose a particular hypothesis as its best guess even though it does not have enough data yet to be sure of the choice.

A solution could be **least-commitment search**.

Version space learning

The hypothesis space can be viewed as a disjunctive sentence

$h_1 \vee h_2 \vee h_3 \vee \ldots \vee h_n$

Hypothesis inconsistent with a new example is removed from the disjunction.

Assuming the original hypothesis space does in fact contain the right answer, the reduced disjunction must still contain the right answer.

The set of hypothesis remaining is called the **version space**.

The version space learning algorithm (also the **candidate elimination** algorithm).

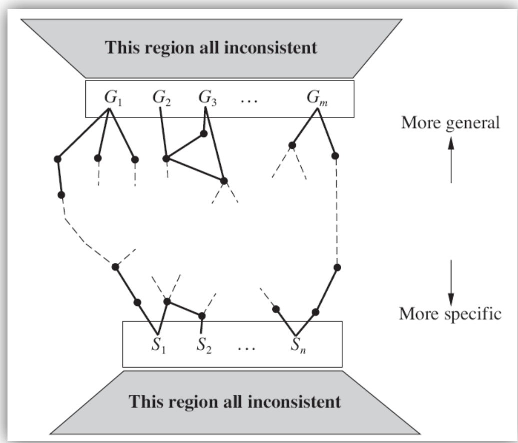
```
function VERSION-SPACE-LEARNING(examples) returns a version space
  local variables: V, the version space: the set of all hypotheses

  V ← the set of all hypotheses
  for each example e in examples do
     if V is not empty then V ← VERSION-SPACE-UPDATE(V, e)
  return V
-----------------------------------------------------------------
function VERSION-SPACE-UPDATE(V, e) returns an updated version space
  V ← {h ∈ V : h is consistent with e}
```

This approach is incremental: one never has to go back and reexamine the old examples

Representation of version space

Hypothesis space is enormous, so how can we possibly write down this enormous disjunction?

We have an ordering of hypothesis space (generalization/specialization) so we can specify boundaries, where each boundary will be a set of hypothesis (a boundary set).



G = a most general boundary
- consistent with all observations so far
- there are no consistent hypotheses that are more general
- initially True

S = a most specific boundary
- consistent with all observations so far
- there are no consistent hypotheses that are more specific
- initially False

Everything in between G-set and S-set is guaranteed to be consistent with the examples and nothing else is consistent.

Version space update

For each new example we update the sets G and S:

- **false positive for S_i**
  - ↳ throw S_i out of the S-set
- **false negative for S_i**
  - ↳ replace S_i in the S-set by all its immediate generalizations
- **false positive for G_i**
  - ↳ replace G_i in the G-set by all its immediate specilaizations
- **false negative for G_i**
  - ↳ throw G_i out of the G-set

The algorithm continues until one of three things happens:

- we have exactly one hypothesis left in the version space
- the version space collapses (either S or G becomes empty)
- we run out of examples and have several hypothesis remaining in the version space
  - the version space represents a disjunction of hypotheses
  - if the hypothesis disagree in classification, one possibility is to take the majority vote

Properties of version space learning

If the **domain contains noise or insufficient attributes** for exact classification, the version space will always collapse.

- to date, no completely successful solution has been found

If we allow unlimited disjunction in the hypothesis space,

- the S-set will always contain a single most-specific hypothesis (the disjunction of the descriptions of positive examples)
- the G-set will contain just the negation of the disjunction of the descriptions of the negative examples
- can be addressed by allowing only limited forms of disjunction by including a **generalization hierarchy** of more general predicates:
  - instead of WaitEstimate(x,30-60) ∨ WaitEstimate(x,>60) we can use LongWait(x)

The pure version space algorithm was first applied in the Meta-DENDRAL system, which was designed to learn rules for predicting how molecules would break into pieces in mass spectrometer.

Inductive logic programming

Inductive logic programming (ILP) combines inductive methods with the power of first-order representations (logic programs).

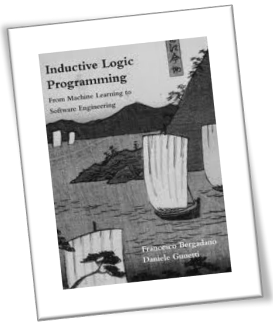
ILP works well with **relationships** between objects, which is hard for attribute-only approaches.

In principle the general knowledge-induction problem is to solve the entailment constraint:

Background ∧ Hypothesis ∧ Descriptions |= Classifications

Two principal approaches to ILP:

- top-down inductive learning methods (system FOIL)
- inductive learning with inverse deduction (system PROGOL)



ILP problem

Background ∧ Hypothesis ∧ Descriptions |= Classifications

- **Examples** are typically given as Prolog facts

```
Father(Philip,Charles), Father(Philip, Anne), …
Mother(Mum,Margaret), Mother(Mum, Elizabeth), …
Married(Diana, Charles), Married(Elizabeth, Philip), …
Male(Philip), Male(Charles), …
Female(Beatrice), Female(Margaret),…
```

- Similarly known **classifications** are given by Prolog facts:

```
Grandparent(Mum,Charles), Gradparent(Elizabeth, Beatrice), …
¬Gradparent(Mum,Harry), ¬Grandparent(Spencer,Peter), …
```

- Possible **hypothesis:**

```
Grandparent(x,y) ⇔ [∃z Mother(x,z) ∧ Mother(z,y)] ∨
                   [∃z Mother(x,z) ∧ Father(z,y)] ∨
                   [∃z Father(x,z) ∧ Mother(z,y)] ∨
                   [∃z Father(x,z) ∧ Father(z,y)]
```

- We can exploit background knowledge:

```
Parent(x,y) ⇔ Mother(x,y) ∨ Father(x,y)
```

- Then we can simplify the hypothesis:

```
Grandparent(x,y) ⇔ [∃z Parent(x,z) ∧ Parent(z,y)]
```

Top-down learning

- Start with a **clause with an empty body**

```
Grandfather(x,y) ←
```

- This clause classifies every example as positive, so it needs to be **specialized**
  - by **adding literals** one at a time to the body

```
Grandfather(x,y) ← Father(x,y)
Grandfather(x,y) ← Parent(x,z)
Grandfather(x,y) ← Father(x,z)
...
```

    - We prefer the specialization that classifies correctly more examples
  - specialize this clause further

```
Grandfather(x,y) ← Father(x,z) ∧ Parent(z,y)
```

  - if background knowledge Parent is not available we may need to add more clauses

```
Grandfather(x,y) ← Father(x,z) ∧ Father(z,y)
Grandfather(x,y) ← Father(x,z) ∧ Mother(z,y)
```

    - each clause covers some positive examples and no negative example

Top-down learning algorithm

```
function FOIL(examples, target) returns a set of Horn clauses
  inputs: examples, set of examples
          target, a literal for the goal predicate
  local variables: clauses, set of clauses, initially empty

  while examples contains positive examples do
     clause ← NEW-CLAUSE(examples, target)
     remove positive examples covered by clause from examples
     add clause to clauses
  return clauses
```

```
function NEW-CLAUSE(examples, target) returns a Horn clause
  local variables: clause, a clause with target as head and an empty body
                   l, a literal to be added to the clause
                   extended_examples, a set of examples with values for new variables

  extended_examples ← examples
  while extended_examples contains negative examples do
     l ← CHOOSE-LITERAL(NEW-LITERALS(clause), extended_examples)
     append l to the body of clause
     extended_examples ← set of examples created by applying EXTEND-EXAMPLE
        to each example in extended_examples
  return clause
```

```
function EXTEND-EXAMPLE(example, literal) returns a set of examples
  if example satisfies literal
     then return the set of examples created by extending example with
        each possible constant value for each new variable in literal
  else return the empty set
```

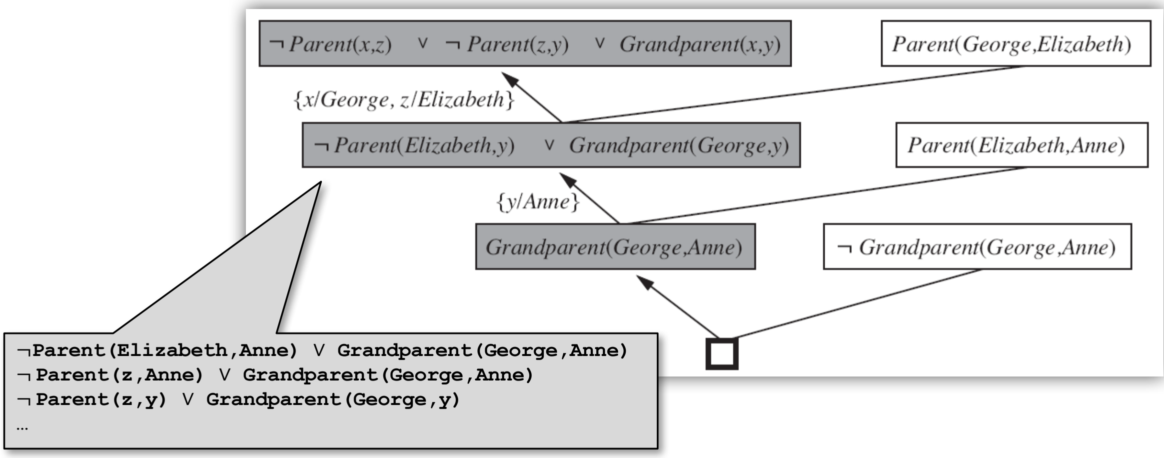
> Build new **clauses** covering positive examples

> **Literals** are chosen from known predicates, equality/inequality literals, and arithmetic comparisons:
> - they have to include a **variable** that is already in clause
> - we can exploit **types** (number, person,…)
> - the choice of literal can be based on **information gain**

System FOIL solved a long sequence of exercises on list-processing functions (for example append, QuickSort).

Background ∧ Hypothesis ∧ Descriptions |= Classifications

- Classical resolution deduces Classifications from Background, Hypothesis, Descriptions.
- We can run the proof backward, find Hypothesis such that the proof goes through:
  - for resolvent C produce C_1 and C_2 (if C_2 is given then produce C_1)



Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz