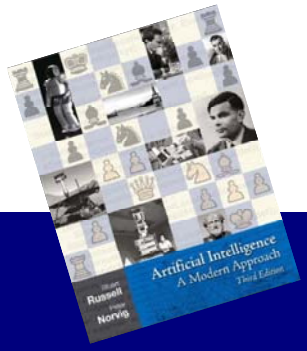


Umělá inteligence II

10

Roman Barták, KTIML

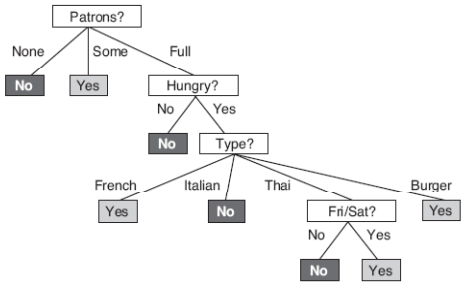
roman.bartak@mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Znalosti v učení

- Umíme se učit funkce vstup → výstup.
- Jedinou dodatečnou znalost, kterou jsme využili, byl tvar hypotézy (prostor hypotéz).
- Můžeme využít již získané znalosti o světě (**background knowledge**)?
- Znalosti budeme nejčastěji vyjadřovat v predikátové logice prvního řádu.
- Metody:
  - metoda nejlepší současné hypotézy
  - prostor verzí
  - induktivní logické programování

Logická formulace učení

- Příklady i hypotézy budou **logické formule**.
- **Příklady**
  - **atributy** budou reprezentovány unárními predikáty
    Alternate(X_1) ∧ ¬Bar(X_1) ∧ ¬Fri/Sat(X_1) ∧ Hungry(X_1) …
  - **klasifikace** příkladu bude také vyjádřena tzv. cílovým predikátem
    WillWait(X_1) nebo ¬WillWait(X_1)
- **Hypotéza** bude mít základní tvar
  ∀x Goal(x) ⇔ C_j(x)
  C_j je tzv. **extenze hypotézy**



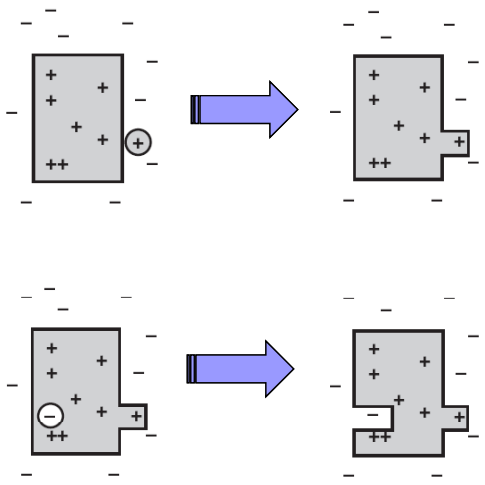
∀r WillWait(r) ⇔ Patrons(r,Some)
∨ (Patrons(r,Full) ∧ Hungry(r) ∧ Type(r,French))
∨ (Patrons(r,Full) ∧ Hungry(r) ∧ Type(r,Thai) ∧ Fri/Sat(r))
∨ (Patrons(r,Full) ∧ Hungry(r) ∧ Type(r,Burger))

Prostor hypotéz

- **Prostor hypotéz** je množina všech hypotéz.
- Učící algoritmus přepokládá, že nějaká hypotéza je správná, tj. platí:
  $h_1 \vee h_2 \vee h_3 \vee \ldots \vee h_n$
- Hypotézy, které nejsou konzistentní s příklady, z disjunkce eliminujeme.
- Dva typy nekonzistence (pro binární klasifikaci) pojmy původně z medicíny
  - **falešně negativní** – hypotéza tvrdí, že příklad je negativní, ale ve skutečnosti je pozitivní
  - **falešně pozitivní** – hypotéza tvrdí, že příklad je pozitivní, ale ve skutečnosti je negativní

Metoda nejlepší hypotézy

- Udržujeme jednu hypotézu (current-best hypothesis) a podle příkladů ji upravujeme
  - příklad je s hypotézou konzistentní
    – hypotézu neměníme
  - falešně negativní
    – hypotézu zobecníme
  - falešně pozitivní
    – hypotézu specializujeme



Metoda nejlepší hypotézy

algoritmus

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
```

Metoda nejlepší hypotézy

specializace a generalizace

Co to přesně znamená zobecnit/specializovat hypotézu?

- Pokud je hypotéza h_1 **zobecněním** hypotézy h_2, potom platí
  $\forall x\ C_2(x) \Rightarrow C_1(x)$
- C_i je typicky konjunkcí predikátů
  - zobecnění uděláme „vypuštěním" podmínek nebo přidáním disjunkce
  - specializaci uděláme přidáním podmínek nebo vypuštěním nějaké disjunkce

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

- Ukázka postupu:
  - první příklad je pozitivní, atribut Alternate(X_1) je pravda, takže nechť je první hypotéza
    h_1: $\forall$x WillWait(x) $\Leftrightarrow$ Alternate(x)
  - druhý příklad je negativní, hypotéza říká pozitivní, máme falešně pozitivní reakci, můžeme specializovat přidáním predikátu
    h_2: $\forall$x WillWait(x) $\Leftrightarrow$ Alternate(x) $\wedge$ Patrons(x,Some)
  - třetí příklad je pozitivní, hypotéza říká negativní, máme falešně negativní reakci, potřebujeme zobecnit, například odstraněním predikátu Alternate(x)
    h_3: $\forall$x WillWait(x) $\Leftrightarrow$ Patrons(x,Some)
  - čtvrtý příklad je pozitivní, hypotéza říká negativní, máme falešně negativní reakci, potřebujeme zobecnit například přidáním disjunkce (odstranit predikát nemůžeme)
    h_3: $\forall$x WillWait(x) $\Leftrightarrow$ Patrons(x,Some) $\vee$ (Patrons(x,Full) $\wedge$ Fri/Sat(x))

Metoda nejlepší hypotézy

vlastnosti

- Po každé modifikaci hypotézy je potřeba **kontrolovat všechny předchozí příklady** (pouze modifikace, které jsou s nimi konzistentní, jsou povolené).
- Při prohledávání často dochází k **množství návratů** (backtracking).
- Prapůvodce problémů – **příliš silný závazek**
  - algoritmus musí v každém kroku vybrat jednu konkrétní hypotézu i když k tomu nemá dostatek informací
- Řešením může být **metoda nejmenších závazků**.

Prostor verzí

- Prostor hypotéz lze reprezentovat disjunkcí $h_1 \vee h_2 \vee h_3 \vee \ldots \vee h_n$
- Pokud je nějaká hypotéza nekonzistentní s příkladem, vyřadíme ji z disjunkce.
- Mezi zbylými hypotézami v disjunkci pořád zůstává ta správná (pokud existuje).
- Obecně **prostor verzí** (version space) je množina možných hypotéz, které jsou konzistentní s dosud zpracovanými příklady.
- Algoritmus **eliminace kandidátů** postupně zmenšuje prostor verzí.

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

$V \leftarrow$ the set of all hypotheses
for each example e in *examples* **do**
  if V is not empty **then** $V \leftarrow$ VERSION-SPACE-UPDATE(V, e)
return V

---

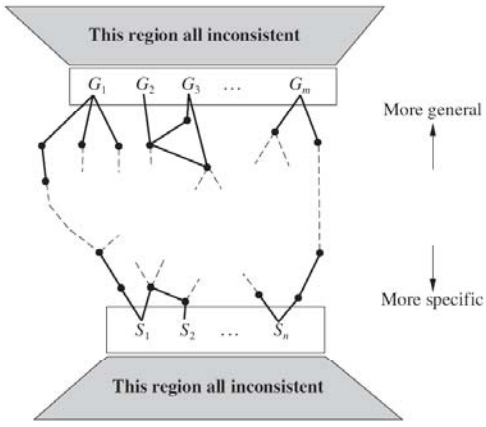
function VERSION-SPACE-UPDATE(V, e) **returns** an updated version space
$V \leftarrow \{h \in V : h \text{ is consistent with } e\}$

- Algoritmus je **inkrementální**
  - po přidání příkladu není potřeba kontrolovat předchozí příklady

Prostor verzí

reprezentace

- Jak reprezentovat velkou (potenciálně nekonečnou) disjunkci?
- Pokud máme k dispozici uspořádání hypotéz (a to máme pomocí zobecnění/specializace), můžeme **prostor verzí reprezentovat dolní a horní mezí**.



G = nejobecnější hypotézy
- konzistentní s dosavadními příklady
- žádná konzistentní hypotéza není obecnější
- na začátku True

S = nejspecifičtější hypotézy
- konzistentní s dosavadními příklady
- žádná konzistentní hypotéza není specifičtější
- na začátku False

- Všechny hypotézy mezi S a G jsou konzistentní s příklady a žádné jiné konzistentní nejsou.

Prostor verzí

update

- Po přidání nového příkladu **upravíme množiny G a S** takto:
  - příklad je **falešně pozitivní pro S_i**
    - vyřadíme S_i z množiny S
  - příklad je **falešně negativní pro S_i**
    - nahradíme S_i v množině S všemi přímými zobecněními
  - příklad je **falešně pozitivní pro G_i**
    - nahradíme G_i v množině G všemi přímými specializacemi
  - příklad je **falešně negativní pro G_i**
    - vyřadíme G_i z množiny G
- **Algoritmus eliminace kandidátů končí** pokud:
  - zůstala jediná hypotéza
  - prostor hypotéz zkolaboval (množina G nebo S je prázdná)
  - vyčerpali jsme všechny příklady
    - prostor verzí potom reprezentuje disjunkci
    - pokud zbylé hypotézy klasifikují různě, můžeme vybrat majoritní odpověď

Prostor verzí

vlastnosti

- **Šum v datech nebo chybějící hodnoty** atributů způsobují kolaps prostoru verzí.
  - zatím nebylo nalezeno obecně fungující řešení
- Pokud povolíme **neomezené disjunkce**, tak
  - S bude disjunkce popisů pozitivních příkladů
  - G bude negace disjunkce popisů negativních příkladů
  - můžeme omezit tvar disjunkce nebo použijeme hierarchii zobecnění predikátů
    - místo WaitEstimate(x,30-60) $\vee$ WaitEstimate(x,>60) použijeme LongWait(x)
- Algoritmus byl použit v systému Meta-DENDRAL pro učení se pravidel předpovídajících, jak se molekuly rozdělí na části při hmotové spektrografii.

ILP

- **Induktivní logické programování** (ILP) kombinuje induktivní učící metody s logickou reprezentací (logickými programy).
- Na rozdíl od technik založených na atributech ILP umí dobře **pracovat s relacemi**.
- V principu se jedná o řešení problému:
  hledání hypotézy použitím známých znalostí Background z příkladů popsaných Descriptions a Classifications, tž.

Background ∧ Hypothesis ∧ Descriptions |= Classifications

Dvě hlavní **metody učení**:

- postup shora-dolů (systém FOIL)
- inverzní rezoluce (systém PROGOL)

ILP problém

Background ∧ Hypothesis ∧ Descriptions |= Classifications

- **Příklady** jsou typicky popsány Prologovskými fakty

```
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),…
```

- Podobně známá **klasifikace** je dána fakty

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

- Možná **hypotéza**

```
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)]
```

- Použijeme-li **znalost**

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

- Potom můžeme hypotézu zjednodušit

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

Metoda shora-dolů

princip

- Začneme **klauzulí s prázdným tělem**

```
Grandfather(x,y) ←
```

- Klasifikuje všechny příklady jako pozitivní, takže ji postupně budeme **specializovat**
  - **přidáním literálu** do těla klauzule

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

    - vybereme tu specializaci, která klasifikuje nejvíce pozitivních a negativních příkladů
  - pokračujeme ve specializaci

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

  - pokud nemáme k dispozici znalost Parent, můžeme získat více klauzulí

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

    - každá klauzule pokrývá některé pozitivní příklady a žádný negativní příklad

Metoda shora-dolů

algoritmus

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*

Postupně stavíme **klauzule** pokrývající pozitivní příklady

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*

Literály se vybírají ze známých predikátů, rovnosti/nerovnosti a aritmetických porovnání
- musí obsahovat **proměnnou**, která už v klauzuli je
- můžeme použít **znalost typů** (číslo, osoba,...)
- volba literálu se dělá na základě **informačního zisku**

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

- Systém FOIL se naučil řadu programů pro práci se seznamy (např. append, QuickSort)

Inverzní rezoluce

- Background $\wedge$ Hypothesis $\wedge$ Descriptions |= Classifications
- Obecná rezoluce tedy musí dokázat Classifications z Background, Hypothesis, Descriptions.
- Můžeme **jít v rezoluci obráceně** a hledat Hypothesis, kterou k důkazu potřebujeme
  - z rezolventy C hledáme C_1 a C_2 (máme-li C_2, hledáme jen C_1)