Rule Learning

- supervised learning
  - rules from decision trees (or the Sequential covering algorithm)
  - PRIM (Bump hunting)
- unsupervised learning
  - association rules
  - version space search for rules
  - Inductive Logic Programming (ILP, MIL)

Association Rules, Market Basket Analysis Application

- For very large datasets, $p \approx 10^4, N \approx 10^8$; in unit ball is the distance to the nearest neighbor ≈ 0.9981.
- We search for **frequent itemsets** (high density areas)
  - example: "the device graph of the home by looking for sets of devices commonly observed on the same IP addresses at the same time"
- We test on feature X_j either equal to a specific value or no restriction at all,
- the value 1 is more important for as than 0,
- We select combinations of items with a higher number of occurences (**support**) than predefined threshold t.
- We select all combinations fulfilling conditions above.
- Categorical variables may be codded by dummy variables in advance (if not too many).
  - **OneHotEncoder** for each class g, a new variable $X_g = [X == g]$ without dropping any value.
- Usually, **sparse** representation (sparse martices) are used.

Apriori Algorithm!

1: **procedure** APRIORI:(X dataset, t threshold for support)
2: $i \leftarrow 1$
3: Generate list of candidates of the length i
4: **while** Candidate set not empty **do**
5: **for** each data sample **do**
6: **for** each candidate **do**
7: **if** all items of candidate appear in the data sample **then**
8: increase the candidate counter by 1
9: **end if**
10: **end for**
11: **end for**
12: $i \leftarrow i + 1$
13: Discard candidates with support less than t.
14: Generate list of candidates of the length i
15: Join any two candidates from previous step having $i - 2$ elements common. *(More pruning possible.)*
16: **end while**
17: **end procedure**

Example: Apriori Algorithm

- $t = 0.2$
- $t * N = 2 = 0.20 * 10$
- Data

a b c e f o
a c g
e i
a c d e g
a c e g l
e j
a b c e f p
a c d
a c e g m
a c e g n

i=1
a=8
b=2
c=8
d=2
e=8
f=2
g=5
i=j=l=o=1
p=m=n=1

i=2
ab=2
ac=8
ad=2
ae=6
af=2
ag=5
bc=2
bd=0
be=2
bf=2
bg=0
cd=2
ce=6
cf=2
cg=5
de=1
df=0
dg=1
ef=2

i=3
abc=2
abd=0
abe=2
abf=2
abg=0
acd=2
ace=6
acf=2
acg=5
ade=1
adf=0
adg=1
aeg=4
...

i=4 ...
abce=2
abcf=2
abef=2
abeg=0
acef=2
aceg=4
adeg=1
aefg=0
...

Properties of the Apriori Algorithm

- Applicable for very large data (with high threshold t).
- The key idea:
  - Only few of 2^K combinations have high support $> t$,
  - **subset of high–support combination has also high support**.
- The number of passes through the data is equal to the size of the longest supported combination. The data do not need to be in memory simultaneously.
- *FPgrowth* algorithm needs only two passes through the data.

Association Rules !

- From each supported itemset $\mathcal{K}$ found by Apriori algorithm we create a list of **association rules**, implications of the form $A \Rightarrow B$ where:
  - A, B are disjoint and $A \cup B = \mathcal{K}$
  - A is called **antecedent**
  - B is called **consequent**.
- Support of the rule $T(A \Rightarrow B)$ is defined as normalized support of the itemset $\mathcal{K}$, that is normalized support of the conjunction $A\&B$.

$$T(\mathcal{K}) = \frac{|data_{\mathcal{K}}|}{|data|}$$

$$T(A \Rightarrow B) = \frac{|data_{A\&B}|}{|data|}$$

Rule Confidence and Lift

There are two important measures for a rule $A \Rightarrow B$:

- **Confidence** (predictability, přesnost)

$$C(A \Rightarrow B) = \frac{T(A \Rightarrow B)}{T(A)}$$

  that is an estimate of $P(B|A)$,
- Support $T(B)$ is an estimate of $P(B)$,
- **Lift** is the ration of confidence and expected precision:

$$L(A \Rightarrow B) = \frac{C(A \Rightarrow B)}{T(B)}$$

  that is an estimate of $\frac{P(A\&B)}{P(A) \cdot P(B)}$.
- **Leverage** is the difference of supports:

$$leverage(A \Rightarrow B) = T(A \Rightarrow B) - T(A) \cdot T(B)$$

- **Conviction** is the ratio:

$$conviction(A \Rightarrow B) = \frac{1 - T(B)}{1 - C(A \Rightarrow B)}.$$

- And many other https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/ .

Association Rule Example

ESL book example:

Association rule 2: Support 13.4%, confidence 80.8%, and lift 2.13.

$$\left[\begin{array}{rcl} \text{language in home} & = & \textit{English} \\ \text{householder status} & = & \textit{own} \\ \text{occupation} & = & \{\textit{professional/managerial}\} \end{array}\right] \Downarrow \text{income} \geq \textit{\$40,000}$$

- $\mathcal{K} = \{$English, own, prof/man, income>\$40000$\}$,
- 13.4% people has all four properties,
- 80.8% of people with {English, own, prof/man} have income$\geq$ \$40000,
- T(income $\geq$ \$40000) = 37.94%, therefore $Lift = 2.13$.

The Goal of Apriori Algorithm !

- Apriori finds all rules with high support.
- Frequently, it finds many of rules.
- We usually select lower threshold c on confidence, that is we select rules with $T(A \Rightarrow B) > t$ and $C(A \Rightarrow B) > c$.
- Conversion of itemsets to rules is usually relatively fast compared to search of itemsets.
- See lispMiner for user interface and a lot of more.
- Python Apriori library:

```
from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.frequent_patterns import fpgrowth,fpmax

from mlxtend.frequent_patterns import hmine

# or another implementation
import efficient_apriori as effapi
```

Demographical Data ESL Example

Feature	Demographic	# Values	Type
1	Sex	2	Categorical
2	Marital status	5	Categorical
3	Age	7	Ordinal
4	Education	6	Ordinal
5	Occupation	9	Categorical
6	Income	9	Ordinal
7	Years in Bay Area	5	Ordinal
8	Dual incomes	3	Categorical
9	Number in household	9	Ordinal
10	Number of children	9	Ordinal
11	Householder status	3	Categorical
12	Type of home	5	Categorical
13	Ethnic classification	8	Categorical
14	Language in home	3	Categorical

Demographical Example – Continuing

- $N = 9409$ questionnaires, the ESL authors selected the 14 questions.
- Preprocessing:
  - `na.omit() remove records with missing values,`
  - `ordinal features cut by median to binary,`
  - `for categorical create dummy variable for each category.`
- Apriori input was matrix 6876×50.
- Output: 6288 association rules
  - with max. 5 elements
  - with support at least 10%.

Negated Literals – Useful, Problematic

Association rule 3: Support 26.5%, confidence 82.8% and lift 2.15.

$$\left[\begin{array}{rcl} \text{language in home} & = & \textit{English} \\ \text{income} & < & \textit{\$40,000} \\ \text{marital status} & = & \textit{not married} \\ \text{number of children} & = & \textit{0} \end{array}\right]$$

$$\Downarrow$$

$$\text{education} \notin \{\textit{college graduate}, \textit{graduate study}\}$$

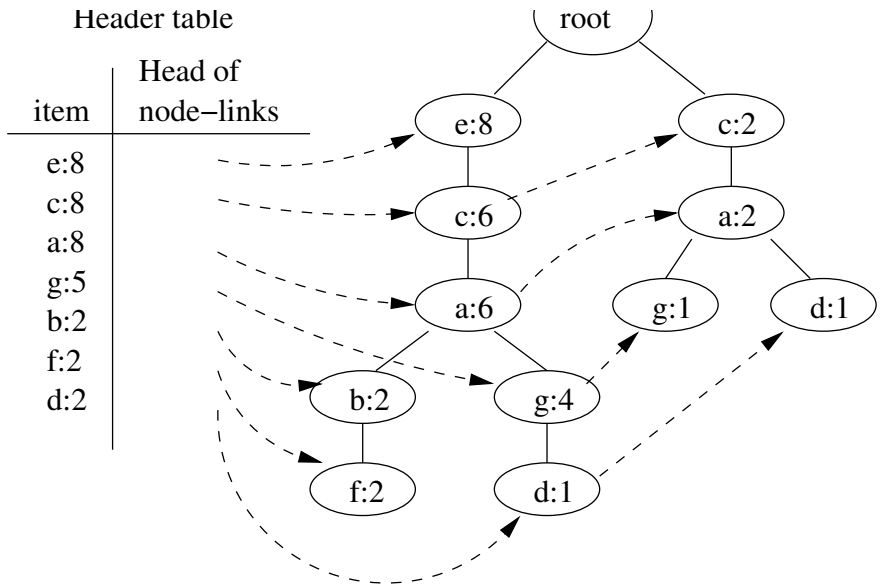
Frequent Pattern–tree data structure

- The number of passes through the data of Apriori is equal to the length of the longest frequent itemset.
- With an internal data structure, we are able to reduce it to 2 passes.
- Build an internal structure called FP-tree.
- Call FP-growth to generate frequent itemsets
  - Each construction of a conditional tree needs 2 pass through the parent tree
  - an optimized version with only 1 pass is presented. (It needs an additional data structure array.)
- FP-max to find maximal itemsets
  - non of immediate supersets is frequent
- FP-close to find close itemsets
  - non of immediate supersets has the same support.

FP–tree

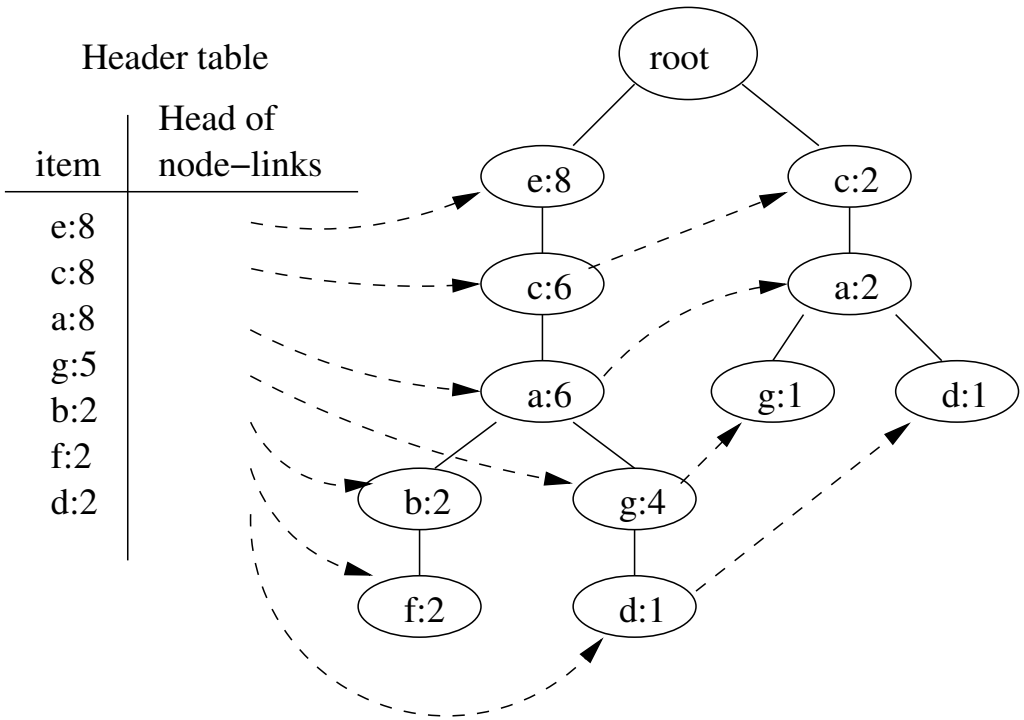
```
1:  procedure FP–TREE:(Data )
2:      Calculate counts of items (singletons)
3:      Create table header ordered by decreasing item count
4:      for each data sample do
5:          order items according to header
6:          insert branch into the tree
7:          increase all counters on the inserted branch
8:      end for
9:      return the tree
10: end procedure
```

Data	ordered
a b c e f o	e c a b f
a c g	c a g
e i	e
a c d e g	e c a g d
a c e g l	e c a g
e j	
a b c e f p	
a c d	
a c e g m	a c e g n



FP-tree

- **FP-tree** contains all the frequency information in the database.
- Principle: If X and Y are two itemsets, the count of itemsets $X \cup Y$ in the database is exactly that of Y in the database restriction to those transactions containing X.



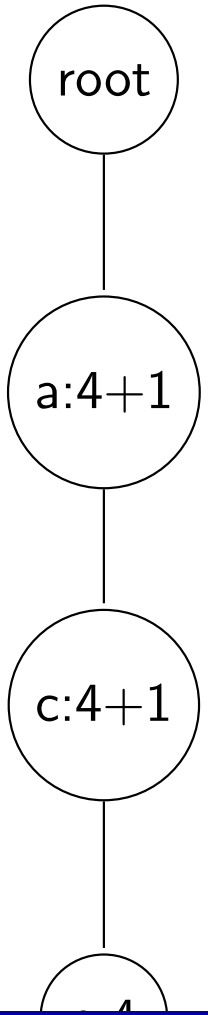
i=3
abc=2
abd=0
abe=2
abf=2
abg=0
acd=2
ace=6
acf=2
acg=5
ade=1
adf=0
adg=1
aeg=4
...

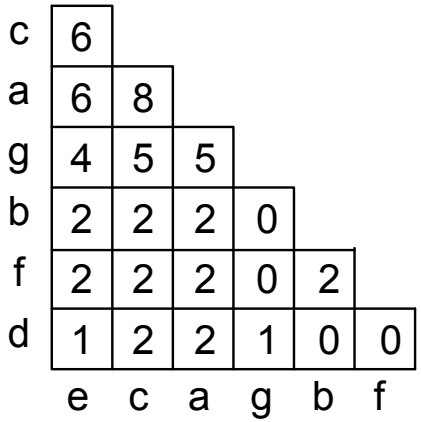
FPgrowth* ×

```
1: procedure FPgrowth*:(T a conditional FP-tree )
2:     if T only contains a single path P then
3:         for each subpath Y of P do
4:             output pattern Y ∪ T.base with
5:                 count = smallest count of nodes in Y
6:         end for
7:     else
8:         for each i in T.header do
9:             Y ← T.base ∪ {i} with i.count
10:            if T.array is not NULL then
11:                construct a new header table for Y's FP-tree from T.array
12:            else
13:                construct a new header table for Y's from T
14:            end if
15:            construct Y's conditional FP-tree T_Y and its array A_Y;
16:            if T_Y ≠ ∅ then
17:                call FPgrowth*(T_Y)
18:            end if
19:        end for
20:    end if
```

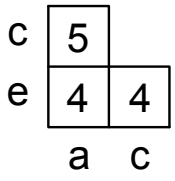
$$t = 2$$

$$T_{\{g\}} = [a:5, c:5, e:4]$$

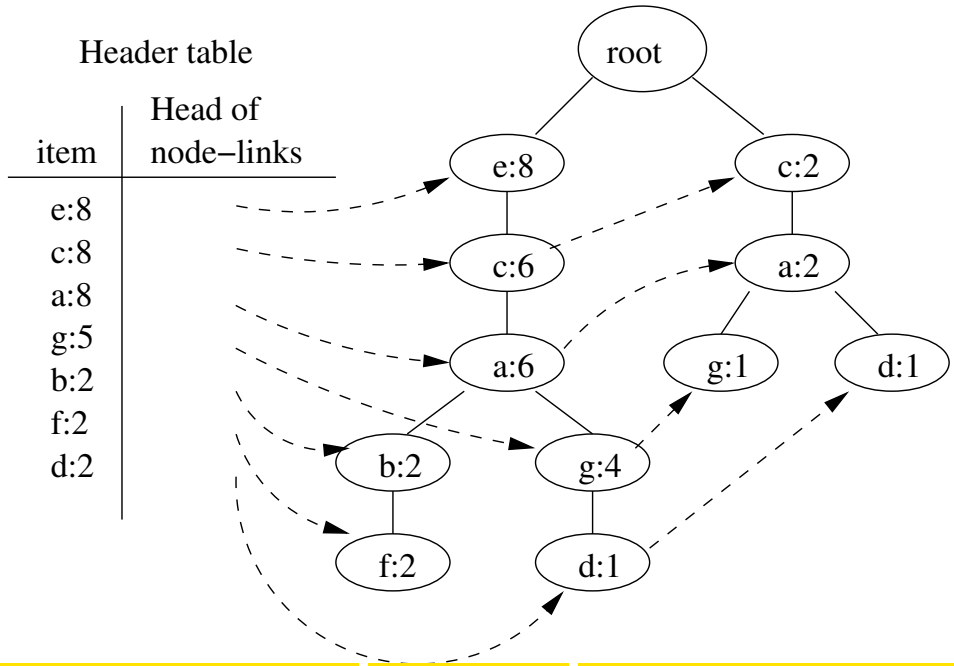




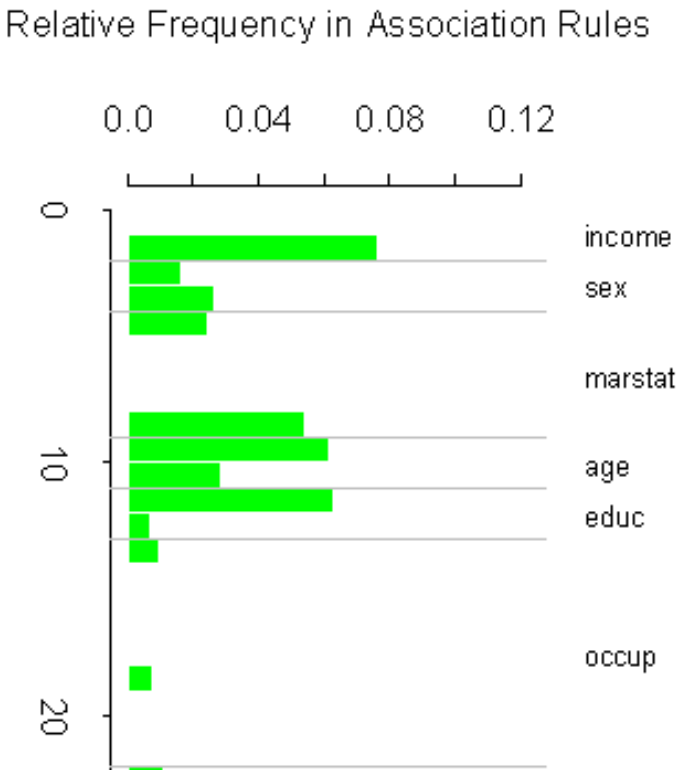
(a) $A_{\emptyset}$

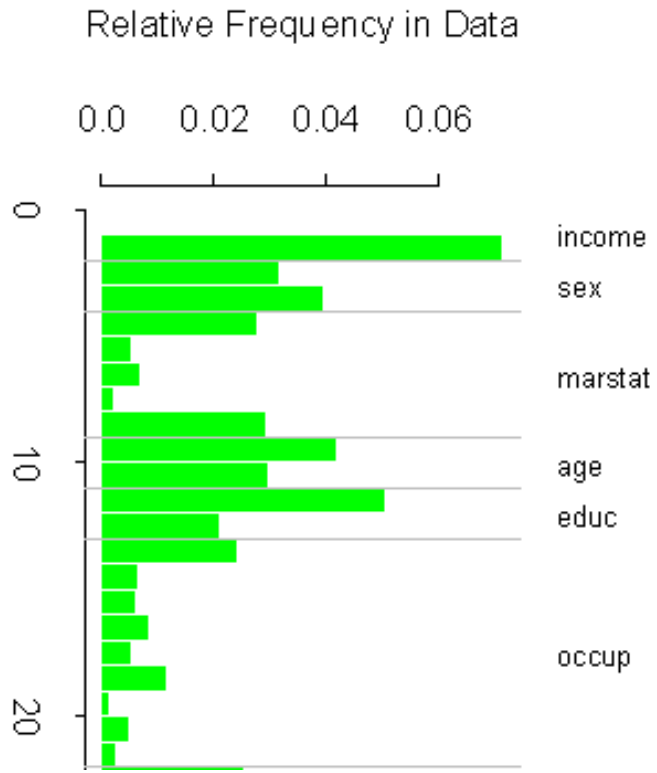


(b) $A_{\{g\}}$

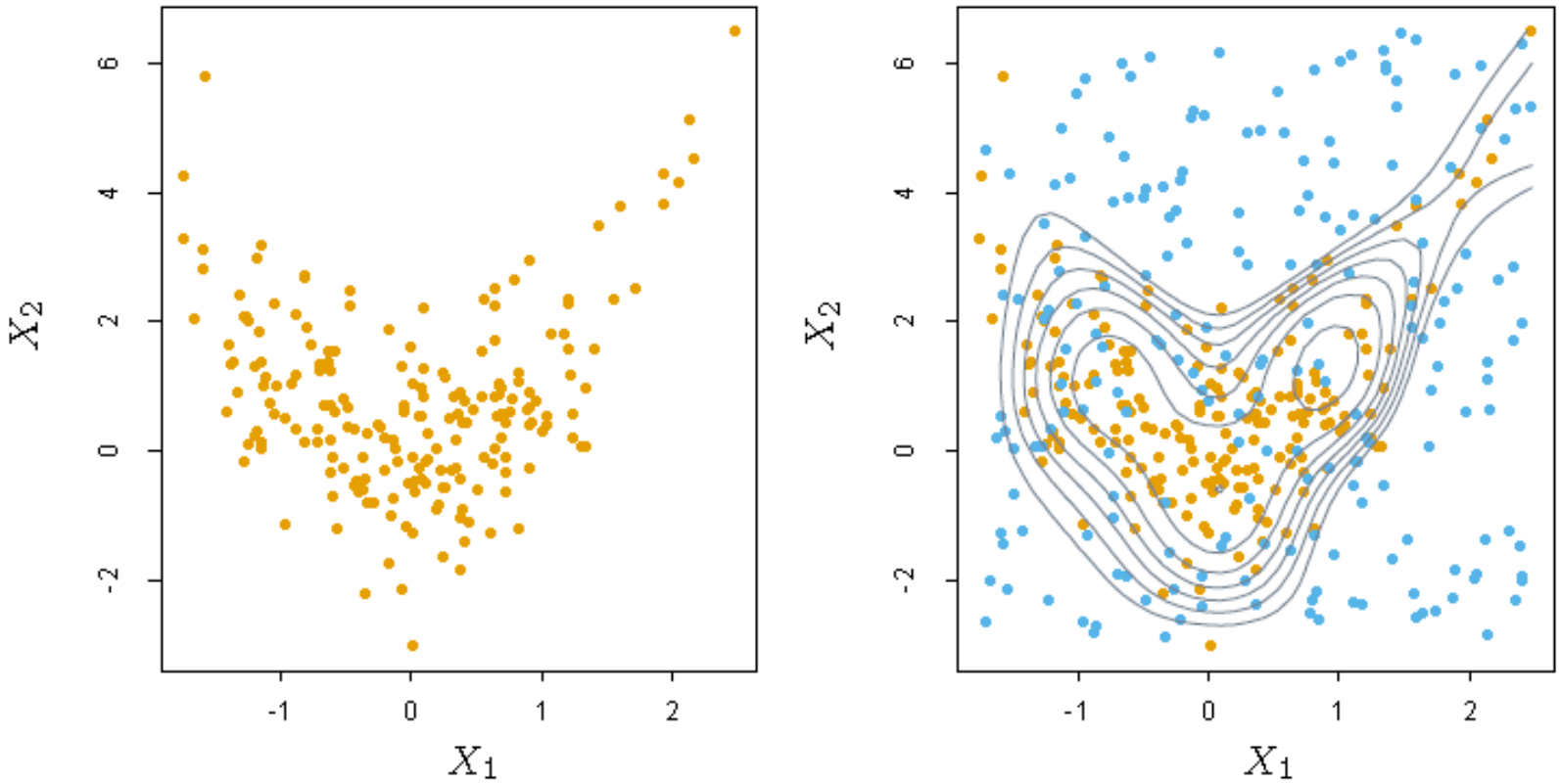


Non–frequent Values Dissapear





Unsupervised Learning as Supervised Learning



- We add an additional attribute Y_G.
- $Y_G = 1$ for all our data.
- We randomly generate a data set of similar size with uniform distribution, set $Y_G = 0$ for these artificial data.
- The task is to separate $Y_G = 1$ and $Y_G = 0$.

Patient Rule Induction Method PRIM = Bump Hunting

- Rule induction method
- We iteratively search regions with the high Y values
  - for each region, a rule is created.
- CART runs of data after aproximately $\log_2(N) - 1$ cuts.
- PRIM can affort $-\frac{\log(N)}{\log(1-\alpha)}$.
  For $N = 128$ data samples and $\alpha = 0.1$ it is 6 and 46 respectively 29, since the number of observations must be a whole number.

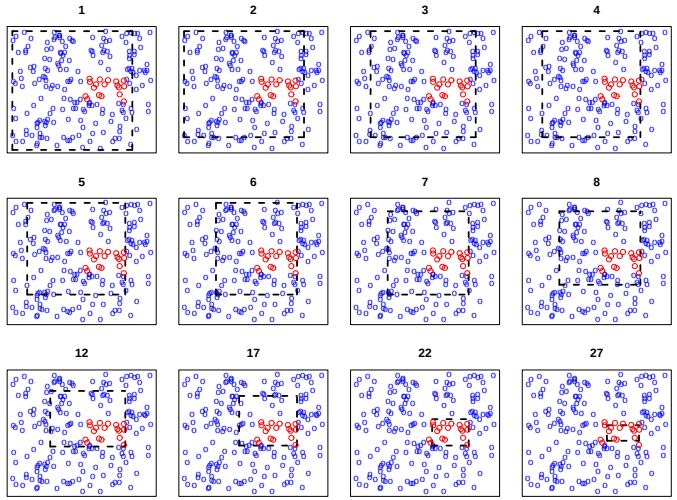


FIGURE 9.7. *Illustration of PRIM algorithm. There are two classes, indicated by the blue (class 0) and red (class 1) points. The procedure starts with a rectangle (broken black lines) surrounding all of the data, and then peels away points along one edge by a prespecified amount in order to maximize the mean of the points remaining in the box. Starting at the top left panel, the sequence of peelings is shown, until a pure red region is isolated in the bottom right panel. The iteration number is indicated at the top of each panel.*

PRIM Patient Rule induction Algorithm

PRIM

- Consider the whole space and all data. Set $\alpha = 0.05$ or 0.10.
- Find X_j and its upper or lower boundary such that the cut of $\alpha \cdot 100\%$ observations leads to the maximal mean of the remaining data.
- Repeat until less then 10 observations left.
- Enlarge the region in any direction that increases the mean value.
- Select the number of regions by the crossvalidation. All regions generated 1-4 are considered.
- Denote the best region B_1.
- Create a rule that describes B_1.
- Remove all data in B_1 from the dataset.
- Repeat 2-5, create B_2 continue until final condition met.

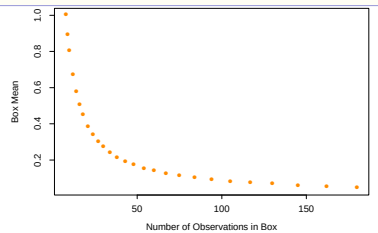


FIGURE 9.8. *Box mean as a function of number o observations in the box.*

Decision Rules from Decision Trees ×

- We can represent a tree as a set of rules
  - one rule for each leaf.
- These rules may be improved by testing each attribute in each rule
  - Has the rule without this test a better precision than with the test?
  - Use validation data
  - May be time consuming.
- These rules are sorted by (decreasing) precision.

Apriori Algorithm Summary

- Unsupervised learning of association rules.
- First, we find **frequent itemsets**, above a threshold t
- Then, we construct rules from them and select
  - high confidence
  - high lift
  - ... .
- The amount of data is expected to be huge;
  - We try to minimize the number of passes through the data
    - the length of the longest frequent itemset for the Apriori algorithm,
    - 2 with the internal structure FP-tree.

Version Space Search

- Version space search is one of the first Machine Learning algorithms.
- For us, an introduction to Inductive Logic Programming.
- Our (Tom Mitchell's) toy data:

Example (Tennis Dataset)

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Overcast	Mild	High	Weak	Yes
D5	Overcast	Mild	High	Strong	Yes
D6	Overcast	Hot	Normal	Weak	Yes
D7	Rain	Mild	High	Strong	No

Version Space Search

- Our **hypothesis** is a conjunction of attribute tests that imply $PlayTennis = yes$.
  - $h = \langle ?, Cold, High, ?, ?, ? \rangle$ represents the hypothesis $Temperature = cold \ \& \ Humidity = high \Rightarrow PlayTennis = yes$.
    - ? is satisfied by any value
    - $\emptyset$ cannot be satisfied
  - For binary attributes, we have $3^{|\#attributes|} + 1$ hypotheses
    - hypotheses with $\emptyset$ are not satisfiable, therefore they are equivalent.
    - We perform a systematic search.
    - The hypothesis space is partially ordered by the subsumption.

Definition (More general, more specific)

- The hypothesis h_g is **more general** than the hypothesis $h_g \succeq h_s$ iff any sample that satisfies h_s satisfies also h_g.
- In the above case, the hypothesis $h_s, h_g \succeq h_s$ is called **more specific that** h_g.

  - $\langle ?, ?, ?, ? \rangle$ is more general than $\langle Sunny, \ldots, Same \rangle$.
  - The most general hypothesis $\langle ?, ?, ?, ? \rangle$ is satisfied by all data.
  - The most specific hypothesis $\langle \emptyset, \ldots \rangle$ is not satisfied by any data.
  - The hypothesis space for a **lattice** partially ordered by the 'more general' relation.

Find–S

- We search for a hypothesis satisfied by all positive examples and no negative example.

Find–S (to be improved)

1: **procedure** FIND-S:(X dataset with the goal attritute *yes*/*no*)
2: $h \leftarrow \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$ # the most specific hypothesis
3: **for** each positive data sample x_i **do**
4: **for** each attribute condition $X_j = x_{i,j}$ in h **do**
5: **if** x_i does not satisfy $X_j = x_{i,j}$ **then**
6: replace the condition by
7: a closest more general condition satisfied by x_i
8: **end if**
9: **end for**
10: **end for**
11: return h
12: **end procedure**

Version Space Search

Example (Tennis Dataset)

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Overcast	Mild	High	Weak	Yes
D5	Overcast	Mild	High	Strong	Yes
D6	Overcast	Hot	Normal	Weak	Yes
D7	Rain	Mild	High	Strong	No

Version Space

- Now we look for all hypotheses consistent with the data.

Definition (Version Space)

- **The version space** for the hypothesis space H and the data X is a subset of H that is consistent with X

$$VS(H, X) = \{h \in H | Consistent(h, X)\}.$$

- The version space is characterized by the most general and the most specific boundary.
- Any hypothesis between these boundaries is consistent with the data.

Definition (General Boundary)

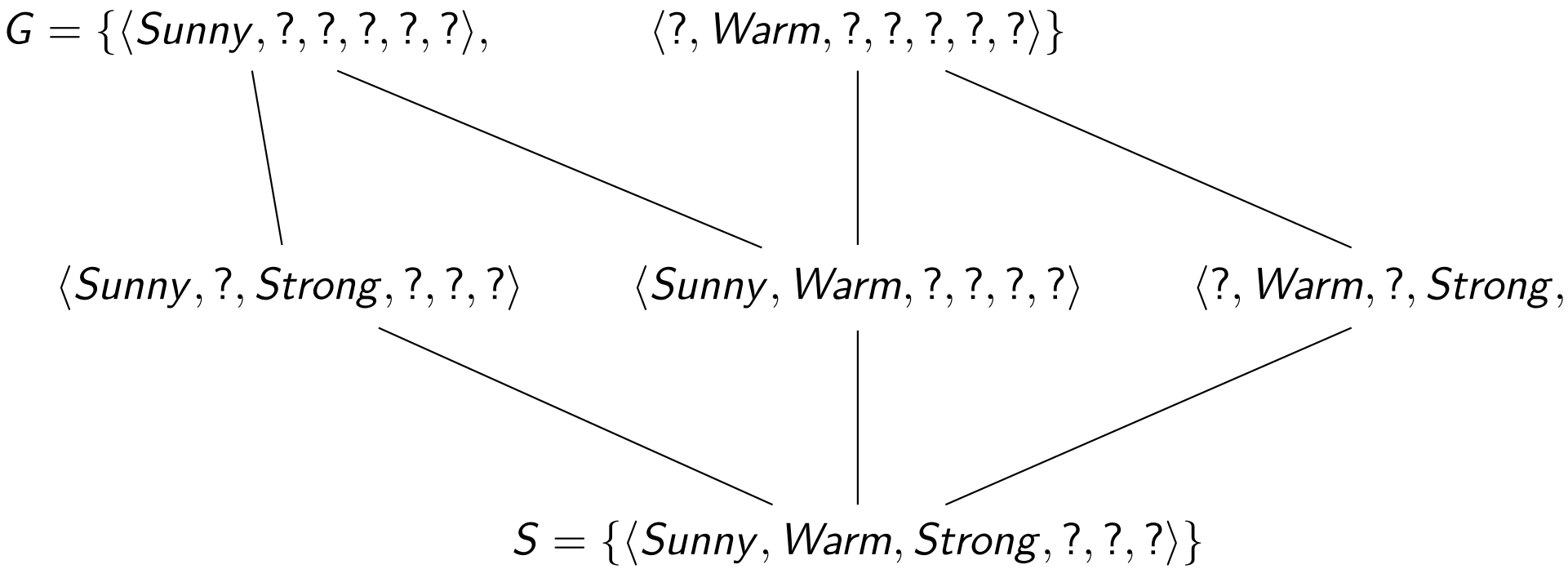
- **The general boundary** for the hypothesis space H and the data X is a set of the most general hypothesis from H that are consistent with X

$$G(H, X) = \{g \in H | Consistent(g, X) \& (\nexists g_1 \in H)[g_1 \succ g \& Consistent(g_1, X)]\}.$$

Definition (Specific Boundary)

- **The specific boundary** for the hypothesis space H and the data X is a set of the most specific hypothesis from H that are consistent with X

$$S(H,X) = \{s \in H | Consistent(s,X) \& (\nexists s_1 \in H)[s \succ s_1 \& Consistent(s_1,X)]\}.$$



- We search for a hypothesis satisfied by all positive examples and none negative example.

1: **procedure** CANDIDATE–ELIMINATION:(X data,the goal att. yes/no)
2: $G \leftarrow \{\langle ?, ?, ?, ? \rangle\}, S \leftarrow \{\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle\}$ # general,specific
3: **for** each data sample x_i **do**
4: **if** x_i is positive **then**
5: remove from G all h inconsistent with x_i
6: **for** each $s \in S$ inconsistent with x_i **do**
7: add to S all minimal generalizations h
8: $Consistent(h, x_i) \& (\exists g \in G)(g \succeq h)$
9: remove from S $\{s | (\exists s_1 \in S)(s \succ s_1)\}$ # not most specific
10: **end for**
11: **else** x_i is negative example
12: remove from S all h inconsistent with x_i
13: **for** each $g \in G$ inconsistent with x_i **do**
14: add to G all minimal specifications h
15: $Consistent(h, X) \& (\exists s \in S)(h \succeq s)$
16: remove from G $\{g | (\exists g_1 \in G)(g_1 \succ g)\}$ # not most gen.
17: **end for**
18: **end if**
19: **end for**
20: return G, S
21: **end procedure**

Literature

- A. Cropper and S. Dumancic. Inductive logic programming at 30: A New Introduction CoRR, abs/2008.07912, 2020.
- S. Muggleton & all.: Meta-interpretive learning: application to grammatical inference, http://www.doc.ic.ac.uk/~shm/FLOC_ILP/Paper03.pdf
- Patsantzis, S., Muggleton, S.H. Top program construction and reduction for polynomial time Meta-Interpretive learning. Mach Learn 110, 755–778 (2021).
- https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html
- https://github.com/stassa/louise
- https://github.com/logic-and-learning-lab/Popper
- `http://www.doc.ic.ac.uk/~shm/FLOC_ILP/Lecture1.1.pdf`
- Olga Štěpánková, Luboš Popelínský: Induktivní logické programování https://www.fi.muni.cz/ popel/lectures/DK-ILP3-fixed.pdf

Predicate Logic

- Recall the predicate logic.
- CNF, DNF the conjunctive and disjunctive normal form
- **clause**: a disjunction of literals $father(X, Y) \vee \neg parent(X, Y) \vee \neg male(X)$
- **Horn clauses** with at most one positive literal, written as a rule
  - **definite clause** $father(X, Y) :- male(X), parent(X, Y).$
  - **fact** - no negative literal $male(adam).$
  - **goal clause** - no positive literal $false :- father(X, bob).$
- **Ground term, clause** - a term, a clause without variables.
- We have our data in the form of a set of clauses B, E^+, E^-,
  - the background knowledge B is a set of (Horn) clauses,
  - the positive and examples E^+, E^- are sets of ground literals (facts).

Example

$$B = \left\{ \begin{array}{c} lego\_builder(alice). \\ enjoys\_lego(A) :- lego\_builder(A). \\ estate\_agent(dave). \\ enjoys\_lego(alice). \\ enjoys\_lego(claire). \end{array} \right\} \quad E^+ = \{happy(alice).\} \quad E^- = \left\{ \begin{array}{c} happy(bob). \\ happy(claire). \\ happy(dave). \end{array} \right\}$$

Substitution, Subsumption

- Clauses ale implicitly generally quantified.
- They should not have a variable with the same name.

Definition (Substitution, Subsumption)

- Given a **substitution** $\theta = \{v_i/t_i\}$ and formula F. $F\theta$ is formed by replacing every variable v_i in F by t_i.
- Substitution θ **unifies** atom A and B in the case $A\theta = B\theta$.
- **Atom A subsumes atom B**, $A \succeq B$, iff there exists a substitution θ such that $A\theta = B$.
- **Clause C subsumes clause D**, $C \succeq D$, iff there exists a substitution θ such that $C\theta \subseteq D$.

Example

- $C_1 = f(A, B) :- head(A, B)$.
- $C_2 = f(X, Y) :- head(X, Y), empty(Y)$.
- C_1 subsumes C_2 since $C_1\theta \subseteq C_2$ with $\theta = \{A/X, B/Y\}$.

Definition (Generalisation)

- Clause C **is more general than** clause D, iff $C \models D$.
- Clause C is more general than clause D with respect to B, iff $B, C \models D$.
  - B is the **background knowledge**.

Example

- Statement A: Daffy Duck can fly. $can\_fly(daffy)$
- Statement B: All ducks can fly. $can\_fly(X) \succeq can\_fly(daffy)$.

Example

- Statement C: Marek lives in London.
- Statement D: Marek lives in England.

- $lives(marek, london)$
- $lives(marek, england)$
- Background knowledge $lives(x, england) : -lives(x, london)$.
- $B, C \models D$, 'C is more general than D with respect to B'.
- $C \succeq D$ with respect to B.
- `http://www.doc.ic.ac.uk/~shm/FLOC_ILP/Lecture1.1.pdf`

ILP general logical setting

Definition (Hypothesis Properies)

The background knowledge B and the hypothesis H should entail E, that is:

Necessity	B	$\not\models$	E^+	we need H
Sufficiency, Completeness	$B \wedge H$	$\models$	E^+	H explains positive examples
Weak consistency	$B \wedge H$	$\not\models$	$\bot$	H does not contradict B
(Strong) **consistency**	$B \wedge H \wedge E^-$	$\not\models$	$\bot$	... neither negative examples

Definition (ILP task)

ILP task is

- Given
  - B background knowledge (logic program)
  - E^+, E^- examples – sets of ground unit clauses
- Given B, E find a logic program H such that is necessary, sufficient and consistent.
- Often, we assume noisy data and accept some errors, but we try to minimize them.

Example

$$B = \left\{ \begin{array}{c} lego\_builder(alice). \\ lego\_builder(bob). \\ estate\_agent(claire). \\ estate\_agent(dave). \\ enjoys\_lego(alice). \\ enjoys\_lego(claire). \end{array} \right\}$$

$$E^+ = \{happy(alice).\}$$

$$E^- = \left\{ \begin{array}{c} happy(bob). \\ happy(claire). \\ happy(dave). \end{array} \right\}$$

Our hypothesis space:

$$\mathcal{H} = \left\{ \begin{array}{c} h_1 : happy(A) : -lego\_builder(A). \\ h_2 : happy(A) : -estate\_agent(A). \\ h_3 : happy(A) : -enjoys\_lego(A). \\ h_4 : happy(A) : -lego\_builder(A), estate\_agent(A). \\ h_5 : happy(A) : -lego\_builder(A), enjoys\_lego(A). \\ h_6 : happy(A) : -estate\_agent(A), enjoys\_lego(A). \end{array} \right\}$$

- $B \cup h_1 \models happy(bob)$ therefore h_1 is inconsistent.
- $B \cup h_2 \not\models happy(alice)$ therefore h_2 is incomplete.
- $B \cup h_3 \models happy(claire)$ therefore h_3 is inconsistent.
- $B \cup h_4 \not\models happy(alice)$ therefore h_4 is incomplete.
- h_5 is both complete and consistent.
- $B \cup h_6 \not\models happy(alice)$ therefore h_1 in incomplete.

Hypothesis Space

- To specify (restrict) the hypothesis space usually mode declarations are used.

Definition (Mode declarations)

Mode declarations denote which literals may appear in the head/body of a rule. A mode declaration is of the form:

$$mode(recall, pred(m_1, m_2, \ldots, m_a))$$

where *recall* is the maximum number of occurrences of the predicate
m_i are the argument types and they may be assigned as input $+$, output $-$, constant $\#$.

Example

```
modeb(2,parent(+person,-person)).
modeh(1,happy(+person)).
modeb(*,member(+list,-element)).
modeb(1,head(+list,-element)).
```

A. Cropper and S. Dumancic. Inductive logic programming at 30: a new introduction.

Non-monotonic reasoning

- In Prolog, there is negation as a failure.

Example

$$Program = \left\{ \begin{array}{c} sunny. \\ happy :- sunny, not\ weekday. \end{array} \right\}$$

- Prolog tries to prove *weekday*.
- It does not prove it, therefore it concludes *happy*.
- With additional knowledge *weekday* some of entailments are not true any more.

Definition (Normal logic program)

Normal logic programs may include negated literals in the body of a clause, e.g.

$$h :- b_1, \ldots, b_n, not\ b_{n+1}, \ldots, not\ b_m.$$

Aleph ILP system (based on Progol)

- Given
  - A set of mode declaration M
  - Background knowledge B in the form of a normal program allows negation, with the semantics negation as a failure
  - Positive E^+ and negative E^- examples as a set of ground facts
- Return: A normal program hypothesis H that:
  - H is consistent with M
  - $\forall e \in E^+$, $H \cup B \models e$ (H is complete)
  - $\forall e \in E^-$, $H \cup B \not\models e$ (H is consistent).

Aleph

1. Select a positive example to generalize.
2. Construct the most specific clause consistent with M that entails the example (the bottom clause).
3. Search for the 'best' clause more general than the bottom clause.
4. Add the clause to the hypothesis and remove all examples covered.
5. If a positive example left, return to step 1.

Bottom Clause Construction

Definition (Bottom clause)

Let H be a clausal hypothesis and C be a clause. The bottom clause $\bot(C)$ is the most specific clause such that:

$$H \cup \bot(C) \models C.$$

- The purpose is to bound the search in the step in 3.
- Without mode declarations, the bottom clause may have infinite cardinality.

Example (Bottom clause)

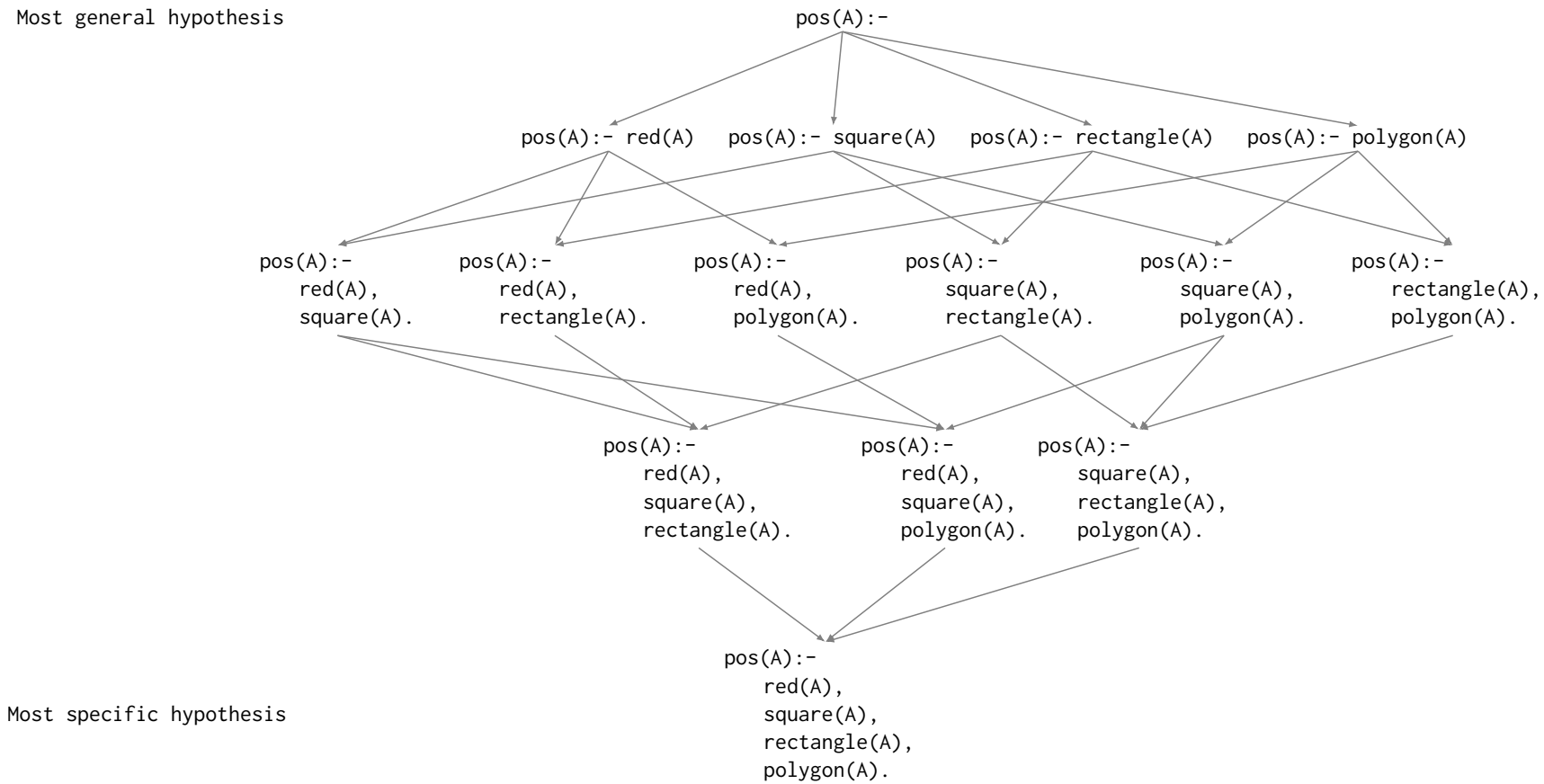
$$M = \left\{ \begin{array}{c} :-modeh(*, pos(+shape)). \\ :-modeb(*, red(+shape)). \\ :-modeb(*, square(+shape)). \\ :-modeb(*, triangle(+shape)). \\ :-modeb(*, polygon(+shape)). \end{array} \right\} \quad B = \left\{ \begin{array}{c} red(s1). \\ blue(s2). \\ square(s1). \\ triange(s2). \\ polygon(A) :- rectangle(A). \\ rectangle(A) :- square(A). \end{array} \right\}$$

Let e be the positive example $pos(s1)$. Then:

$$\bot(e) = pos(A) :- red(A), square(A), rectangle(A), polygon(A).$$

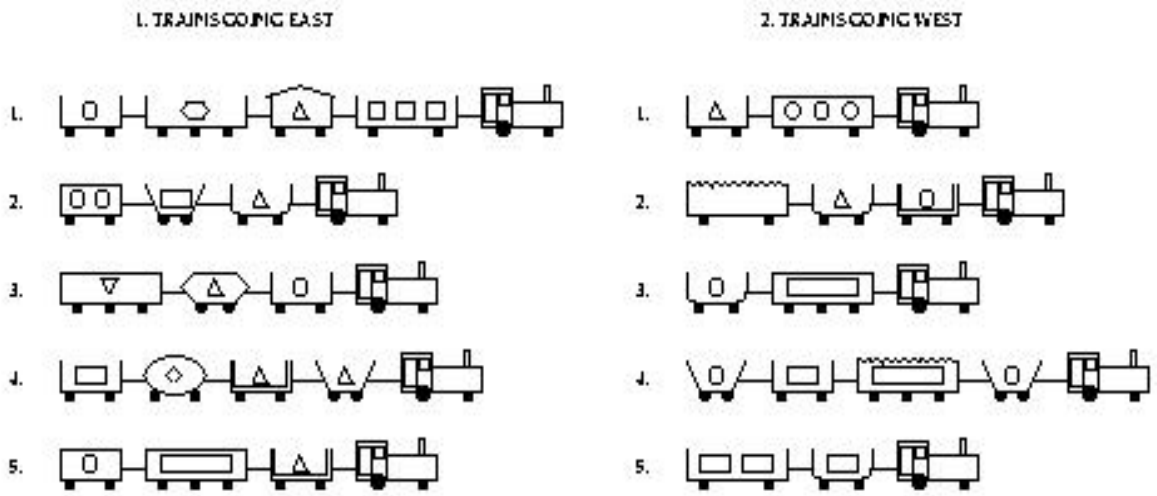
Clause Search

- Aleph performs a bounded-breadth-first search to enumerate the shorter clauses before the longer ones.
- The search is bounded by several parameters (max. clause size, max. proof depth).



Example (Luboš Popelínský & all.)

East-West Trains (1)



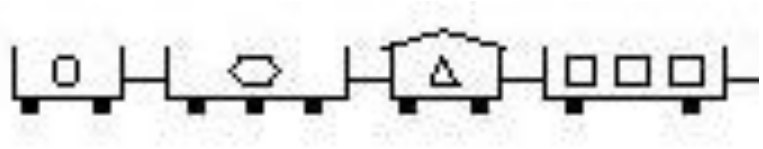
East-West Trains (2)

eastbound(east1).
eastbound(east2).
eastbound(east3).
eastbound(east4).
eastbound(east5).

eastbound(west6).
eastbound(west7).
eastbound(west8).

% eastbound train 1
short(car_12). closed(car_12).
long(car_11). open_car(car_11).
…
shape(car_11,rectangle). shape(car_12,rectangle).
…
load(car_11,rectangle,3). load(car_12,triangle,1).
…
wheels(car_11,2). wheels(car_12,2).

East-West Trains (4)



[bottom clause][literals] [25][saturation time] [0.01]

```
eastbound(A) :-
   has_car(A,B), has_car(A,C), has_car(A,D), has_car(A,E),
   short(B), short(D), closed(D), long(C), long(E), open_car(B), open_car(C), open_car(E),
   shape(B,rectangle), shape(C,rectangle), shape(D,rectangle), shape(E,rectangle),
   wheels(B,2), wheels(C,3), wheels(D,2), wheels(E,2),
   load(B,circle,1), load(C,hexagon,1), load(D,triangle,1), load(E,rectangle,3).
```

```
:- modeh(1,eastbound(+train)).
:- modeb(*,has_car(+train,-car)).
:- modeb(1,short(+car)).
:- modeb(1,load(+car,#shape,#int)).
...
```

```
[reduce]
eastbound(A).                          [5/5]
eastbound(A) :- has_car(A,B).          [5/5]
eastbound(A) :-
      has_car(A,B), short(B).          [5/5]
…
eastbound(A) :-
    has_car(A,B),wheels(B,3).          [3/1]
eastbound(A) :-
    has_car(A,B), closed(B).           [5/2]
eastbound(A) :- has_car(A,B),
    load(B,triangle,1).                [5/2]
```

Aleph 2, Popper, FlexFringe

- Aleph default evaluation function is **coverage** defined as $P - N$,
  - P is the number of positive examples covered by the clause
  - N is the number of negative examples covered by the clause
  - that means it accepts some noise.
- It starts from the most general one $pos(A) :-$.
- It tries to specialize the clause
  - by adding literals to the body of it, which it selects from the bottom clause
- or by instantiating variables.
- Each specialization is called **refinement**.
- Aleph Advantages
  - one Prolog file, easy to download and use.
    - https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html
  - It has good empirical performance.
  - Allows numerical reasoning, user defined cost functions, handles noisy data.
- Aleph Disadvantages
  - It has many parameters to tune.
  - It struggles to learn recursive programs and optimal programs
    - since it learns only a single clause a time.

Metagol

- Given
  - A set of metarules M
  - Background knowledge B in the form of a normal program
  - Positive E^+ and negative E^- examples as a set of facts (atoms).
- Return: A definite program hypothesis H that:
  - H is consistent with M
  - $\forall e \in E^+$, $H \cup B \models e$ (H is complete)
  - $\forall e \in E^-$, $H \cup B \not\models e$ (H is consistent)
  - $\forall h \in H$, $\exists m \in M$ such that $h = m\theta$
    - where θ is a substitution that grounds all the existentially quantified variables in m.

Example (Metarule)

- An example is the **chain** metarule $P(A, B) \leftarrow Q(A, C), R(C, B)$
- that allows Metagol to induce programs such as

$$f(A, B) \quad :- \quad tail(A, C), tail(C, B).$$
$$grandparent(A, B) \quad :- \quad parent(A, C), parent(C, B).$$

Metagol

- Metagol is a form of ILP besed on a Prolog meta-interpreter.

Metagol

1. Select a positive example to generalize.
   - If none exists, test the hypothesis on the negative examples.
     - If the hypothesis does not entail any negative example stop and return the hypothesis.
     - otherwise backtrack to a choice point at step 2 and continue.
2. Try to prove the atom by:
   - using given BK or an already induced clauses
   - unifying the atom with the head of a metarule
   - binding the variables in the metarule to symbols in the predicate and constant signatures
   - save the substitution
   - try to prove the body of the metarule by treating the body atoms as examples and applying step 2 to them.

Recursion

- Metagol can learn recursive programs.

Example (Reachability)

Consider learning the concept of *reachability* in a graph. Without recursion, with the maximal depth 4 we could learn:

$$
\begin{aligned}
reachable(A,B) &:- edge(A,B).\\
reachable(A,B) &:- edge(A,C), edge(C,B).\\
reachable(A,B) &:- edge(A,C), edge(C,D), edge(D,B).\\
reachable(A,B) &:- edge(A,C), edge(C,D), edge(D,E), edge(E,B).
\end{aligned}
$$

With recursion, we can learn:

$$
\begin{aligned}
reachable(A,B) &:- edge(A,B).\\
reachable(A,B) &:- edge(A,C), reachable(C,B).
\end{aligned}
$$

Iterative deepening

iterative deepening

- Metagol uses iterative deepening to search for hypotheses.
  - at depth $d = 1$, at most one metasub.
  - at iteration d, it introduces $d - 1$ new predicate symbols and is allowed to use d clauses.

Metagol Example

Example (Kinship example)

$$B = \left\{ \begin{array}{c} mother(ann, amy).mother(ann, andy). \\ mother(amy, amelia), mother(amy, bob). \\ mother(linda, gavin). \\ father(steve, amy).father(steve, andy). \\ father(andy, sponegebob).father(gavin, amelia). \end{array} \right\}$$

$$metarule(ident, [P, Q], [P, A, B], [[Q, A, B]]).$$
$$metarule(chain, [P, Q, R], [P, A, B], [[Q, A, C], [R, C, B]]).$$

$$E^+ = \left\{ \begin{array}{c} grandparent(ann, amelia). \\ grandparent(steve, amelia). \\ grandparent(ann, spongebob). \\ grandparent(linda, amelia). \end{array} \right\}$$
$$E^- = \{grandparent(amy, amelia).\}$$

Tracing Metagol

- It select the first example to generalize $grandparent(ann, amelia)$.
- It tries to prove it from BK and induced clauses. It fails.
- Metagol tries to use the first metarule:

$$grandparent(ann, amelia) : -Q(ann, amelia).$$

  stores $sub(ident, [grandparent, Q])$
- and tries to unify Q, but fails.
- Metagol tries to use the second metarule:

$$grandparent(ann, amelia) : -Q(ann, C), R(C, amelia).$$

  stores $sub(chain, [grandparent, Q, R])$
- and recursively tries to prove $Q(ann, C)$ and $R(C, amelia)$.
- It succeedes with the metasum $sub(chain, [grandparent, mother, mother])$
- and induces the first clause;

$$grandparent(A, B) : -mother(A, C), mother(C, B).$$

Metagol Trace 2

- Then, it select the second example to generalize $grandparent(steve, amelia)$.
- It tries to prove it from BK and induced clauses. It fails.
- Metagol can again use the second metarule with another substitution: stores $sub(chain, [grandparent, father, mother])$
- and induces the second clause;

$$grandparent(A, B) :- father(A, C), mother(C, B).$$

- Given no bound on the program size, the Metagol would prove the other two examples the same way and form the program:

$$\begin{aligned}
grandparent(A, B) &:- mother(A, C), mother(C, B).\\
grandparent(A, B) &:- father(A, C), mother(C, B).\\
grandparent(A, B) &:- father(A, C), father(C, B).\\
grandparent(A, B) &:- mother(A, C), father(C, B).
\end{aligned}$$

With predicate invention, it learns:

$$\begin{aligned}
grandparent(A, B) &:- grandparent_1(A, C), grandparent_1(C, B).\\
grandparent_1(A, B) &:- father(A, B).\\
grandparent_1(A, B) &:- mother(A, B).
\end{aligned}$$

Tail Recursive Metarule

Example (Tail Recursive Metarule)

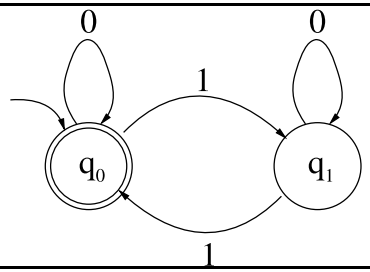
- An example is the **tail recursive** metarule $P(A, B) \leftarrow Q(A, C), P(C, B)$
- Metagol can also learn mutually recursive programs, such:

$$
\begin{aligned}
&even(0).\\
even(A) \quad &:- \quad successor(A, B), even\_1(B).\\
even\_1(A) \quad &:- \quad successor(A, B), even(B).
\end{aligned}
$$

We even do not have to provide the concept of an odd number. We can let the Metagol to invent such predicate (*even*_1).

Automata Example

(a)

Finite acceptor	Production rules	Definite Clause Grammar (DCG)
	$q_0 \rightarrow$ $q_0 \rightarrow 0\ q_0$ $q_0 \rightarrow 1\ q_1$ $q_1 \rightarrow 0\ q_1$ $q_1 \rightarrow 1\ q_0$	$q_0([], [])\ \leftarrow$ $q_0([0\|A], B)\ \leftarrow q_0(A, B)$ $q_0([1\|A], B)\ \leftarrow q_1(A, B)$ $q_1([0\|A], B)\ \leftarrow q_1(A, B)$ $q_1([1\|A], B)\ \leftarrow q_0(A, B)$

(b)

E^+	E^-	Meta-interpreter	Ground facts
λ 0 00 11 000 011 101	1 01 10 001 010 100 111	$parse(S) \leftarrow parse(q0, S, []).$ $parse(Q, [], []) \leftarrow acceptor(Q).$ $parse(Q, [C\|X], Y) \leftarrow$ $delta1(Q, C, P),$ $parse(P, X, Y).$	$acceptor(q0) \leftarrow$ $delta1(q0, 0, q0) \leftarrow$ $delta1(q0, 1, q1) \leftarrow$ $delta1(q1, 0, q1) \leftarrow$ $delta1(q1, 1, q0) \leftarrow$

Fig. 1 (**a**) Parity acceptor with associated production rules, DCG; (**b**) positive examples (E^+) and negative examples (E^-), Meta-interpreter and ground facts representing the Parity grammar

http://www.doc.ic.ac.uk/~shm/FLOC_ILP/Paper03.pdf

Louise Example Grammar Learning

Example (Module)

```
:-module(anbn, [background_knowledge/2
   ,metarules/2
   ,positive_example/2
   ,negative_example/2
   ,a/2
   ,b/2
   ]).
```

Example (Background knowledge)

```
background_knowledge(s/2,[a/2,b/2]).
a([a|T],T).
b([b|T],T).
```

Example (Metarules)

metarules(s/2,[chain]).
% (Chain) $\exists$.P,Q,R $\forall$.x,y,z: P(x,y)$\leftarrow$ Q(x,z),R(z,y)

Louise Example Continued

Example (Positive Examples)

```
positive_example(s/2,E):-
  member(E, [%s([a,b],[])
   s([a,a,b,b],[])
   ]).
```

Example (Negative Examples)

```
negative_example(s/2,E):-
  member(E,[s([a,a],[])
   ,s([b,b],[])
   ,s([a,a,b],[])
   ,s([a,b,b],[])
   ]).
```

Example (Parameter Tuning)

```
:- auxiliaries:set_configuration_option(clause_limit, [3]).
:- auxiliaries:set_configuration_option(max_invented, [1]).
:- auxiliaries:set_configuration_option(reduction, [none]).
```

Louise Example Continued 2

Example (Learned Program)

```
'$1'(A,B):-a(A,C),a(C,B).
'$1'(A,B):-a(A,C),s(C,B).
'$1'(A,B):-b(A,C),b(C,B).
'$1'(A,B):-s(A,C),b(C,B).
s(A,B):-'$1'(A,C),'$1'(C,B).
s(A,B):-'$1'(A,C),b(C,B).
s(A,B):-a(A,C),'$1'(C,B).
s(A,B):-a(A,C),b(C,B).
true.
```

Example (Learned Program)

```
?- auxiliaries:set_configuration_option(unfold_invented, [true]).
?-make.
?- learn(s/2).
s(A,B):-a(A,C),b(C,B).
s(A,B):-a(A,C),s(C,D),b(D,B).
true.
```

ASPAL algorithm

- ASPAL uses Answer Set Programming.
- ASP program can have one, many, or none models (answer sets).
- Computation in ASP is the process of finding models.
- We may specify the range of the number of clauses from a set beeing true. $0\{sunny., weekday., happy(A) : -lego\_builder(A)\}3$
- We may specify an evaluation function to optimize (like to minimize the number of 'true' clauses, e.g. the size of the hypothesis.

ASPAL

- Generate all possible rules consistent with the given mode declarations.

  Assign each rule a unique identifier and add an guessable atom in each rule.
- Use an ASP solver to find a minimal subset of the rules

  by formulating the problem as an ASP optimization problem.

Example (ASPAL)

$$B = \left\{ \begin{array}{c} bird(alice). \\ bird(betty). \\ can(alice, fly). \\ can(betty, swim). \\ ability(fly). \\ ability(swim). \end{array} \right\} \qquad M = \left\{ \begin{array}{c} modeh(1, penguin(+bird)). \\ modeb(1, bird(+bird)). \\ modeb(*, not\ can(+bird, \#ability)). \end{array} \right\}$$

$$\begin{array}{lcl} E^+ & = & \{penguin(betty).\} \\ E^- & = & \{penguin(alice).\} \end{array}$$

Given the modes, the possible rules are:

$$\begin{array}{lcl} penguin(X) & :- & bird(X). \\ penguin(X) & :- & bird(X), not\ can(X, swim). \\ penguin(X) & :- & bird(X), not\ can(X, fly). \\ penguin(X) & :- & bird(X), not\ can(X, swim), not\ can(X, fly). \end{array}$$

ASPAL replaces constants and adds extra literal:

$$\begin{array}{lcl} penguin(X) & :- & bird(X), rule(r1). \\ penguin(X) & :- & bird(X), not\ can(X, C1), rule(r2, C1). \\ penguin(X) & :- & bird(X), not\ can(X, C1), not\ can(X, C2), rule(r3, C1, C2). \end{array}$$

ASPAL passes to an ASP solver:

$bird(alice).$

$bird(betty).$

$can(alice, fly).$

$can(betty, swim).$

$ability(fly).$

$ability(swim).$

$penguin(X) :- bird(X), rule(r1).$

$penguin(X) :- bird(X), not\ can(X, C1), rule(r2, C1).$

$penguin(X) :- bird(X), not\ can(X, C1), not\ can(X, C2), rule(r3, C1, C2).$

$0\{rule(r1), rule(r2, fly), rule(r2, swim), rule(r3, fly, swim)\}4$

$goal :- penguin(betty), not\ penguin(alice).$

$:- not\ goal.$

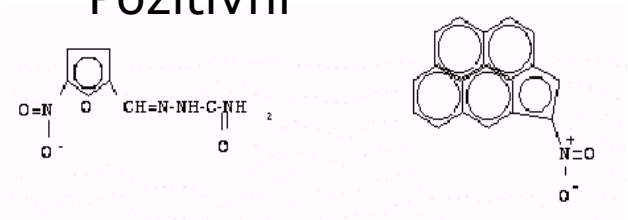
- The answer is: $rule(r2, c(fly))$
- Which is translated to a program:

$$penguin(A) :- bird(A), not\ can(A, fly).$$

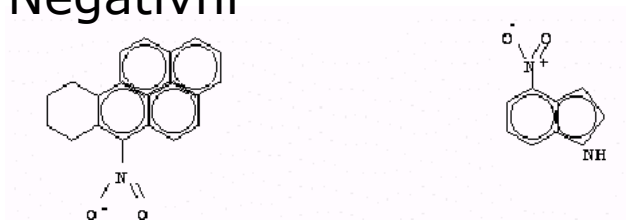
ILP applications

- Bioinformatics
  - ILP can make predictions based on the (sub)structured biological data.
  - Predict mutagenic activity of molecules and alert the causes of chemical cancers
  - learning protein folding signatures.

Pozitivní



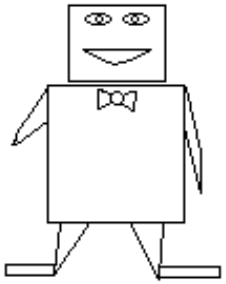
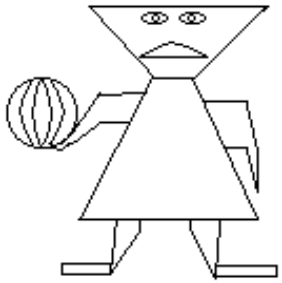
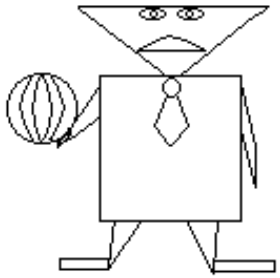
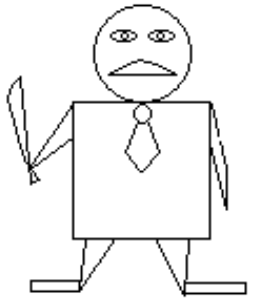
Negativní



- Robot scientist.
  - BK knowledge represents the relationship between protein-coding sequences, enzymes, and metbolites in pathway.
  - Automatically generates hypotheses, runs experiments, and iterates results.
- Games
  - Sokoban
  - Bridge
  - Checkers.

Computer Game Application (Luboš Popelínský & all.)

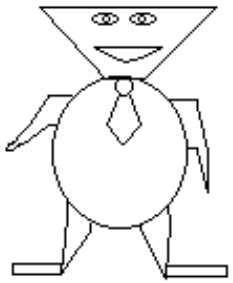
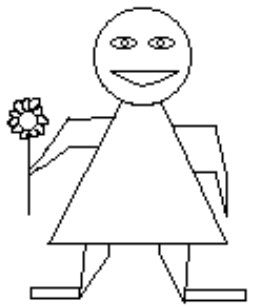
? Differentiate the friendly agent (4 left) from the enemies (4 right).

přátelští

nepřátelští



List of topics

1. Linear, ridge, lasso regression, k-nearest neighbors, overfitting, curse of dimensionality, (LARS)
2. Splines - the base, natural splines, smoothing splines; kernel smoothing: kernel average, Epanechnikov kernel.
3. Logistic regression, Linear discriminant analysis, (generalized additive models)
4. Train/test error and data split, square error, 0-1, cross-entropy, AIC, BIC, cross-validation, one-leave-out CV, biased estimate example
5. decision trees, information gain/entropy/gini, CART α pruning
6. random forest (+bagging), OOB error, Variable importance, boosting (Adaboost and gradient boosting), stacking, (MARS),
7. Bayesian learning: MAP, ML hypothesis, Bayesian optimal prediction !formulas!, EM algorithm
8. Clustering: k-means, Silhouette, k-medoids, hierarchical
9. Apriori algorithm, Association rules, support, confidence, lift
10. Inductive logic programming basic: hypothesis space search, background knowledge, necessity, sufficiency and consistency of a hypothesis, Aleph
11. Undirected graphical models, Graphical Lasso procedure, (deviance, MRF)
12. Gaussian processes: estimation of the function and its variance !figures, ideas!
13. Support Vector Machines (optimal separating hyperplane, slacks, kernels).

Table of Contens