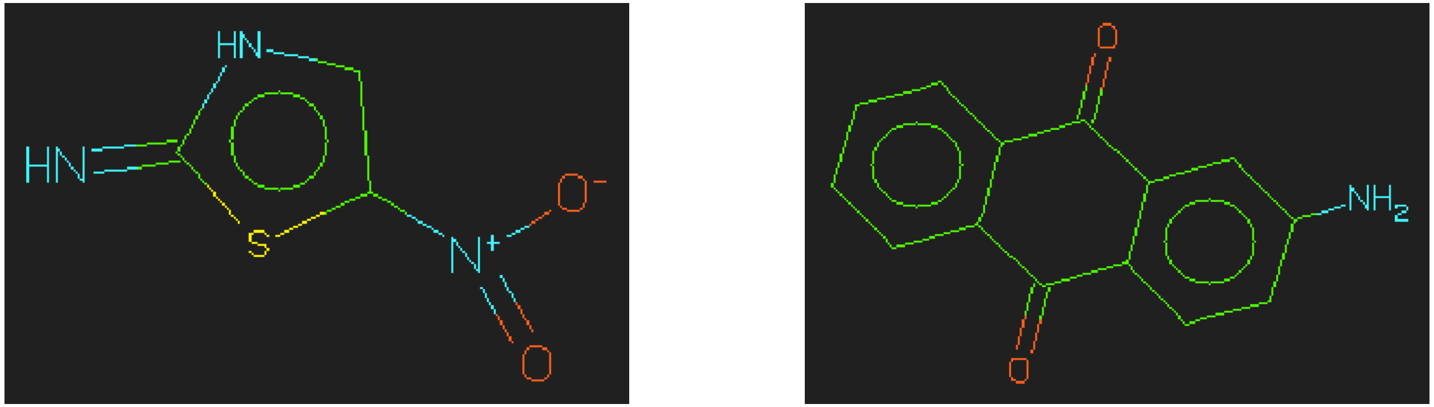
Relational Data Mining



Univ.-Lektor Dr.techn. Alexander K. Seewald
Österreichisches Forschungsinstitut
für Artificial Intelligence

alex@seewald.at / alex.seewald.at

Relational Data Mining

Basics

- Data Mining on multiple tables within a relational database *without* collapsing into a single tabular representation.
- Uses techniques from *Inductive Logic Programming* (ILP) which learns (logical) programs from training data.

Pros

- Can accomodate background knowledge in various forms
- Lends itself to comprehensible models in complex domains

Cons

- Highly inefficient on large datasets
- Propositional approaches are competitive in performance

Motivation: A Complex Learning Task*

Goals

- automatic discovery of systematic relationships between chemical structure of substances and their carcinogenicity (*structure-activity relationships*, *SARs*)
- help reduce the number of animal experiments needed to test toxicity/carcinogenicity of new chemical or pharmaceutical substances.

Data

International databases with thousands of complex chemical compounds and information about toxicity gained from animal experiments

- global properties of compounds
- exact molecular structure
- results of laboratory tests

* Project performed at the Austrian Research Institute for Artificial Intelligence (ÖFAI), Vienna in cooperation with Department of Cancer Research / Tumor Biology, University of Vienna; funded by the Austrian Federal Ministry for Education, Science, and Culture.

Tabular data...

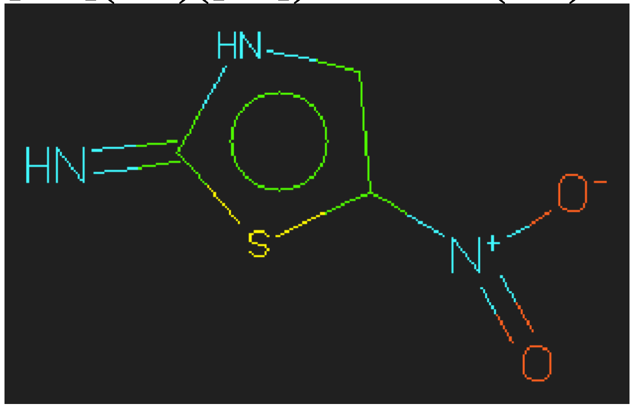
ID	subst	sp.	sex	strain	route	tissue	tumor	xpot.	xprt.	**td50**	pval
d120	bzt	m	f	b6c	0	lun	a/a	104	106	**3890**	0
d120	bzt	m	f	b6c	0	lun	a/c	104	106	**4390**	0
d120	bzt	m	f	b6c	0	nci	neg	104	106	**18900**	0.8
d120	bzt	m	f	b6c	0	liv	hpa	104	106	**1.00E+31**	1
d120	bzt	m	f	b6c	0	lun	a/a	104	106	**3890**	0
d120	bzt	m	m	b6c	0	nci	neg	104	106	**1.00E+31**	1
d120	bzt	m	m	b6c	0	liv	hpa	104	106	**1.00E+31**	1
d120	bzt	m	m	b6c	0	lun	a/a	104	106	**1.00E+31**	1
d120	bzt	r	f	f34	0	bra	gln	78	105	**20300**	0.3
d120	bzt	r	f	f34	0	nci	neg	78	105	**1.00E+31**	1
d120	bzt	r	f	f34	0	liv	hpa	78	105	**10100**	0.1
d120	bzt	r	m	f34	0	bra	gln	78	104	**1.00E+31**	1
d120	bzt	r	m	f34	0	liv	nnd	78	104	**2720**	0
d120	bzt	r	m	f34	0	nci	neg	78	104	**1.00E+31**	1
d120	bzt	r	m	f34	0	liv	hpa	78	104	**2720**	0

Tumorigenic dose (TD50)

...and structural data

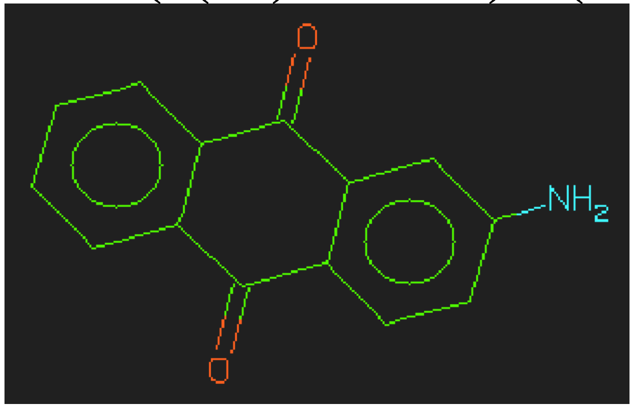
Molecular structure of d120

[N+](=O)([O-])C1=CNC(=N)S1



Molecular structure of d68

O=C1c3c(C(=O)c2c1cccc2)ccc(c3)N



Hard but not impossible to represent as a table. However, relational representations (~ first-order logic) are more compact and better comprehensible.

alex@seewald.at / alex.seewald.at

First-order logic (Prolog): Basic Notions (1)

Constant

- Symbol representing a specific object, value, or property (*3.14*, *joe*, *v4_xy*, *true*)
- Written as a number or lower-case string

Variable

- Represents a general argument of a predicate (*X*, *Name*, _)
- Can be bound to a constant (same constant within each context)
- Written as a string starting with an upper-case letter or _

Predicate (relation)

- Is applied to arguments: constants or variables (*is_pi/1*, *taller/2*, *gcd/3*, ...)
- Written as string starting with a lower-case letter, followed by a *(*
- Expresses specific properties of or relations between its arguments, can be of arbitrary arity (number of arguments N≥0), denoted via */N*

Term

- Predicate with arguments of constants and/or variables (*is_pi(3.14)*, *taller(joe,peter)*, *taller(X,Y)*, ...)

First-order logic (Prolog): Basic Notions (2)

Literal

- A term (e.g., *p(X,Y)*) or its negation (e.g., \+ *p(X,Y)*)

Rule (Horn clause)

- A term of the form H :- L_1 , L_2 , L_3 , ...
- Where H (the *head*) is a term and the L_i (the *conditions*) are literals
- The conditions are conjoined by *and* (,)
- A call to a rule succeeds if, after unifying the call with the head of the rule, all the conditions succeed (which may involve additional (recursive) rule calls)

Example: *stronger(X,Y) :- taller(X,Y), heavy(X).*

Theory (Program)

- A set of rules with the same head

Interpretation: a call to a theory succeeds if at least one of the rules succeeds

⇒ Theory is a disjunction of rules, e.g.

stronger(X,Y) :- taller(X,Y), heavy(X).
stronger(X,Y) :- same_size(X,Y), heavier(X,Y).
stronger(X,Y) :- name(X,arnold).

The ILP Learning Task

Given...

- *Training examples*: positive and negative ground examples of the target concept
- *Background knowledge*: a set of literals that can be used as conditions in the program, and their definition ⇒ (additional) description of the training examples
- *Target relation*: variabilised head of searched-for program.

Find...

- *Theory*: a definition of the target relation (= a set of first-order classification rules that are consistent with the training examples)

Separate-And-Conquer Rule Learning

Learning Rule Sets: Sequential Set Covering Algorithm

```
procedure LearnRules($c$,$TD$)
$Rules$ := {}; $Pos$ := {$x_i \in TD \mid y_i = c$};
$Neg$ := {$x_i \in TD \mid y_i \neq c$};
repeat
    $Rule$ := FindBestRule($c$,$Pos$,$Neg$);
    $Pos$ := $Pos$ \ Covers($Rule$,$Pos$);
    $Neg$ := $Neg$ \ Covers($Rule$,$Neg$);
    $Rules$ := $Rules \cup Rule$;
until RuleSetStopCrit($Rules$,$Pos$,$Neg$);
return ($Rules$);
```

Simplest stopping criterion 1:

RuleSetStopCrit = true if all positive examples are covered by the current rule set, i.e. *Pos* = {}

How to learn a theory?

Sequential Set Coverting, i.e. Rule Learning!

RuleSet → Theory/Program

Rule → Rule (Horn clause)

```
procedure FindBestRule($c$,$Pos$,$Neg$)
$Rules$ := { $\Rightarrow c$ }    // ← create default rule
while not RuleStopCrit($Rule$,$Pos$,$Neg$) do
    $Rule$ := $Rule \cup$ FindBestCondition($Pos$,$Neg$);
    $Pos$ := Covers($Rule$,$Pos$);
    $Neg$ := Covers($Rule$,$Neg$);
endwhile; return ($Rule$);
```

Simplest stopping criterion 2:

RuleStopCrit = true if the rule covers no more negative examples, i.e., *Neg* = {}. Such a rule is called **consistent**.

FindBestCondition → FindBestLiteral

(i.e. test all literals and choose best via heuristic,

usually Weighted Information Gain)

A rule **covers** an example if the conditions of the rule match the attribute values of the example. Rules always predict the *positive* class.

FOIL (First-Order Inductive Learner)

Characteristics

- *Instance Space*: Prolog ground facts
- *Hypothesis Space*: Recursive function-free Prolog programs (sets of classification rules in the form of Horn clauses)
- *Search direction*: top-down (general-to-specific) search
- *Search strategy*: sequential covering, hill-climbing
- *Search heuristic*: weighted information gain
- *Enhancements*: determinate literals, noise handling (pruning), check points, implicit definition of negative examples (closed world assumption),...

Implementation: http://www.rulequest.com/Personal/foil6.sh

Example

Training examples:

+: *father(christopher,arthur).*
+: *father(christopher,victoria).*
-: *father(penelope,arthur).*
-: *father(christopher,penelope).*

Background knowledge:

male(christopher)
male(arthur)

female(victoria)
female(penelope)

parent(christopher,arthur).
parent(christopher,victoria).
parent(penelope,arthur).
parent(penelope,victoria).

Target relation: *father/2*

Running FOIL...

```
| ?- foil(father(A,B),Clauses).
2 positive and 2 negative instances.
  male(A):    Gain: 0.83 (2+/1-)
\+male(A):    Gain: 0.00 (0+/1-)
  male(B):    Gain: 0.00 (1+/1-)
\+male(B):    Gain: 0.00 (1+/1-)
  female(A): Gain: 0.00 (0+/1-)
\+female(A): Gain: 0.83 (2+/1-)
  female(B): Gain: 0.00 (1+/1-)
\+female(B): Gain: 0.00 (1+/1-)
  male(A):    Gain: 0.00 (2+/1-)
\+male(A):    Gain: 0.00 (0+/0-)
  male(B):    Gain: 0.58 (1+/0-)
\+male(B):    Gain:-0.42 (1+/1-)
  female(A): Gain: 0.00 (0+/0-)
\+female(A): Gain: 0.00 (2+/1-)
  female(B): Gain:-0.42 (1+/1-)
\+female(B): Gain: 0.58 (1+/0-)
Chose literal male(A).
  Gain: 0.83 (2+/1-).
```

```
  parent(A,A): Gain: 0.00 (0+/0-)
\+parent(A,A): Gain: 0.00 (2+/2-)
  parent(A,B): Gain: 0.83 (2+/1-)
\+parent(A,B): Gain: 0.00 (0+/1-)
  parent(B,A): Gain: 0.00 (0+/0-)
\+parent(B,A): Gain: 0.00 (2+/2-)
  parent(B,B): Gain: 0.00 (0+/0-)
\+parent(B,B): Gain: 0.00 (2+/2-)
  parent(A,A): Gain: 0.00 (0+/0-)
\+parent(A,A): Gain: 0.00 (2+/1-)
  parent(A,B): Gain: 1.17 (2+/0-)
\+parent(A,B): Gain: 0.00 (0+/1-)
  parent(B,A): Gain: 0.00 (0+/0-)
\+parent(B,A): Gain: 0.00 (2+/1-)
  parent(B,B): Gain: 0.00 (0+/0-)
\+parent(B,B): Gain: 0.00 (2+/1-)
Chose literal parent(A,B).
  Gain: 1.17 (2+/0-).
Found clause:
  father(A,B):-male(A),parent(A,B)
```

Introducing new variables

It is sometimes necessary to introduce new variables in the body of clause

Example: *grandfather(A,B) :- father(A,C), parent(C,B).*

Method

- Introduce literal with new variable(s)
- Candidate literal must contain at least one old variable!
- When counting how many positive and negative instances match the new literal (for computing info gain), the new variable can be bound to all values that appear in background knowledge for a fixed binding of the old variables.

Counting covered instances

- Count whether there is a binding that yields a positive example or not (counting instances)
- Count number of possible bindings that yield a positive example (counting proofs)

New variables: An Example

Current rule: *grandfather(A,B) :-*

Possible conditions (literals) to extend the clause..

male(A) \+*male(A)*
male(B) \+*male(B)*
female(A) \+*female(A)*
female(B) \+*female(B)*

parent(A,A) \+*parent(A,A)*
parent(A,B) \+*parent(A,B)*
parent(B,A) \+*parent(B,A)*
parent(B,B) \+*parent(B,B)*
parent(A,X) \+*parent(A,X)*
parent(X,A) \+*parent(X,A)*
parent(B,X) \+*parent(B,X)*
parent(X,B) \+*parent(X,B)*

~~*parent(X,X)* \+*parent(X,X)*~~
~~*parent(X,Y)* \+*parent(X,Y)*~~

Determinate (zero-gain) literals

Determinate literals: A literal is determinate if it...

- contains one or more new variable(s)
- the new variable(s) can be bound in exactly one way for each positive example
- the new variable(s) can be bound in at most one way for each negative example

Example: *age(Person,Years)* is determinate (each person has exactly one age)

Determinate literals usually have low or even **zero information gain** (do not help to discriminate between the pos. and neg. examples)

⇒ Determinate literals would not be selected by common search heuristics

BUT: they may be important because they introduce new variables that are needed in further tests, which then do discriminate.

Example: *baby(Person) :- age(Person,X),* $X < 1$.

Extensional vs. Intensional Knowledge

Extensional background knowledge

Explicit listing of (ground Prolog) facts about objects of the domain

- **Target relation**: *grandfather(A,B).*
- **Training examples**: *grandfather(paul,mary).*
  grandfather(paul,peter).
- **Background knowledge**: facts about persons, such as...
  parent(paul,sally).
  parent(sally,mary).
  parent(paul,chris).

Intensional background knowledge

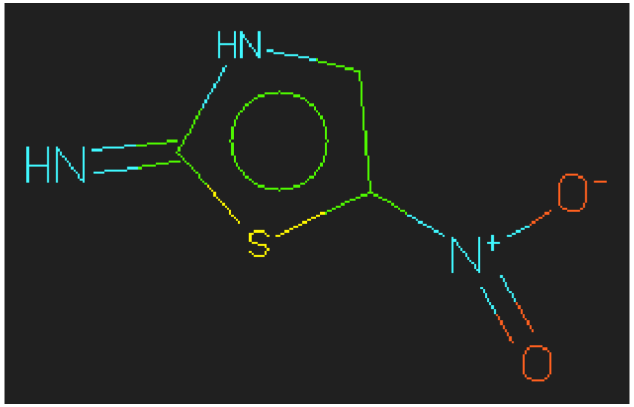
Definition of properties and relations by means of general Prolog predicates (arbitrarily complex programs, might have been learned!)

parent(X,Y) :- father(X,Y).	*father(paul,sally).*
parent(X,Y) :- mother(X,Y).	*father(paul,chris).*
	mother(sally,mary).

© Alexander K. Seewald
alex@seewald.at / alex.seewald.at

Extensional Background Knowledge

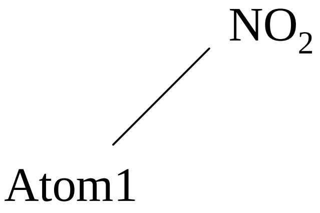
```
% chemical structure of our substances:
% Smiles code [N+](=O)([O-])C1=CNC(=N)S1
%
% atm(SubstID, AtomID, Element, Charge, XVal)
atm(d120,d120_1,n,0,0).
atm(d120,d120_2,n,0,0).
atm(d120,d120_3,c,0,0).
atm(d120,d120_4_1,h,0,0).
atm(d120,d120_4,c,0,0).
atm(d120,d120_5_1,h,0,0).
atm(d120,d120_5,c,0,0).
....
% bond(SubstID, Atom1, Atom2, BondType).
bond(d120,d120_1,d120_2,2).
bond(d120,d120_1,d120_9,1).
bond(d120,d120_2,d120_3,1).
bond(d120,d120_3,d120_4,7).
bond(d120,d120_3,d120_8,7).
bond(d120,d120_4,d120_4_1,1).
bond(d120,d120_4,d120_5,7).
bond(d120,d120_5,d120_5_1,1).
bond(d120,d120_5,d120_6,7).
....
```



Describe all atom types & positions (*atm/5*) plus their connections (*bond/4*) explicitly.

Intensional Background Knowledge

```
% a nitro group:
nitro(Drug,[Atom0,Atom1,Atom2,Atom3]):-
    atm(Drug,Atom0,n,_,_),
    sym_bond(Drug,Atom0,Atom1,1),
    sym_bond(Drug,Atom0,Atom2,_),
    Atom2 \== Atom1,
    atm(Drug,Atom2,o,_,_),
    sym_bond(Drug,Atom0,Atom3,_),
    Atom3 \== Atom1,
    Atom3 @> Atom2,
    atm(Drug,Atom3,o,_,_).
....
....
....
....
% alcohol
alcohol(Drug,[Atom0,Atom1,Atom2]):-
    atm(Drug,Atom0,o,_,_),
    sym_bond(Drug,Atom0,Atom1,1),
    atm(Drug,Atom1,h,_,_),
    sym_bond(Drug,Atom0,Atom2,1),
    atm(Drug,Atom2,Type1,_,_),
    Type1 \== h,
    aryl(Drug,Atom2,Atom0).
....
....
....
....
```



NO_2

Atom1

recursive reference to another intensional chemical concept

alex@seewald.at / alex.seewald.at

Learned Theory

```
carc(A, true) :-
    ion_pot(A, B), gteq(B, 11.09158),
    log_p(A, C), gteq(C, 0.4121).
carc(A, false) :-
    ion_pot(A, B), gteq(B, 11.09158),
    \+(log_p(A,C),gteq(C,0.4121)).
carc(A, true) :-
    \+(ion_pot(A,B), gteq(B,11.09158)),
    alkyl_halide(A,_), m_weight(A, C),
    gteq(C, 545.546).
carc(A, true) :-
    \+(ion_pot(A,B),gteq(B,11.09158)),
    \+(m_weight(A,C),gteq(C,545.546)),
    atm(A, _, n, _, _), ell_vol(A, D),
    gteq(D, 258.688049).
carc(A, false) :-
    \+(ion_pot(A,B),gteq(B,11.09158)),
    \+(m_weight(A,C),gteq(C,545.546)),
    atm(A, _, n, _, _),
    \+ (ell_vol(A,D),gteq(D,258.688049)).
```

```
carc(A, true) :-
  \+(m_weight(A,C),gteq(C,545.546)),
  \+atm(A,_,n,_,_), atm(A, _, c, _, _),
  surf_area(A, D), gteq(D, 249.525818).
carc(A, true) :-
  \+(ion_pot(A,B),gteq(B,11.09158)),
  \+(m_weight(A,C),gteq(C,545.546)),
  \+atm(A,_,n,_,_), atm(A, _, c, _, _),
  \+(surf_area(A,D),gteq(D,249.525818)),
  t_energy(A, E), gteq(E, -1528.75049).
carc(A, false) :-
  \+(ion_pot(A,B),gteq(B,11.09158)),
  \+(m_weight(A,C),gteq(C,545.546)),
  \+atm(A,_,n,_,_), atm(A, _, c, _, _),
  \+(surf_area(A,D),gteq(D,249.525818)),
  \+(t_energy(A,E),gteq(E,-1528.75049)).
....
....
....
....
```