# Data Mining – Relational Learning

Andreas Karwath Jörg Wicker

Johannes Gutenberg University Mainz

January 26, 2017

# Acknowledgments

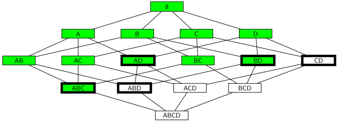
- Slides partially taken from
  - Stefan Kramer
  - Tom Mitchel
  - Johannes Frnkranz

Section 1

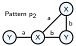
# Introduction

# The Story so far ...

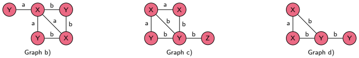
- Item mining




- Graph mining
  - Subgraph pattern $p_2$:

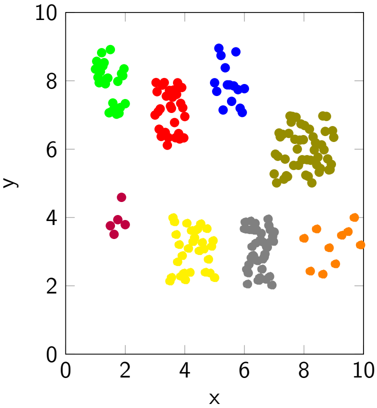



  - Database $D$ consisting of graphs $b$), $c$), and $d$):



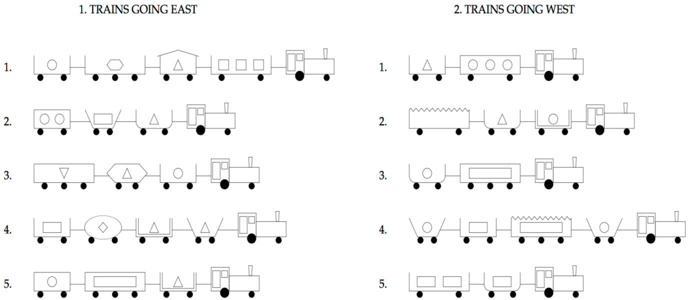

  - support($p_2$, $D$) = 2

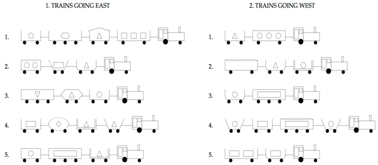
- Clustering



# Relational Learning Introduction I

- Offers a versatile way of describing data and patterns (pattern language)
- Can be applied for item set mining as well as for graph mining or …
- just for all sorts of relational problems.
- But, what are relational problems?
- For example the train example from last weeks outlook:





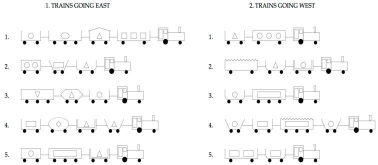

# Relational Learning Introduction II

- So what rule can describe the difference between east and west trains?
- One possible explanation (or so-called hypothesis):
  Trains go east if they possess a short car with a roof.
- The rule is fine for humans, but how can one model this using computers?
- One way would be to "flatten" it:

| #Ex. | $C1_{shape}$ | $C1_{axes}$ | $C1_{roof}$ | $C1_{\#objects}$ | $C1_{objectShape}$ | ... | Class |
|---|---|---|---|---|---|---|---|
| 1 | rectangle | 2 | open | 3 | square | ... | east |
| 2 | bucket | 2 | open | 1 | triangle | ... | east |
| ... | ... | ... | ... | ... | ... | ... | ... |

- However, this is not ideal:
  - change in the order of cars
  - different shape in the cargo
  - varying number of cars
  - ...

# Relational Learning Introduction III




- A more "natural" way of describing these examples is in language able to cater for relational data: e.g. Prolog
- Example:

```
east(e1).
has_car(e1,e1c1).
shape(e1c1,rectangle).
length(e1c1,short).
roof(e1c1,open).
carry(e1c1,e1c1o1,square).
carry(e1c1,e1c1o2,square).
carry(e1c1,e1c1o3,square).
...
```

# Quick Prolog Recap

- Variables: `X` , `Y`, `A`, `B`, `Train`, `Car`
- Terms: `square`, `t1`, `1`, `[1,2,3]`
- Predicates: `east/1`, `shape/1`, `carry/3`

- Facts:

```
shape(e1c1,rectangle).
carry(e1c1,e1c1o1,square).
```

- Rules:

```
east(Train) :- has_car(Train,Car), length(Car,short), roof(Car,open).
```

# Logical reasoning: Deduction

- From rules to facts

$$B \cup T \vdash E$$

- $B$: Background knowledge
- $T$: Theory
- $E$: Examples

| $B$ | $\cup$ | $T$ | $\vdash$ | $E$ |
|---|---|---|---|---|

```
mother(penelope,victoria).
mother(penelope,arthur).
father(christopher,victoria).
father(christopher,arthur).
```

```
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

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

# Logical reasoning: Induction

- From facts to rules

$$B \cup E \vdash T$$

- $B$: Background knowledge
- $T$: Theory
- $E$: Examples

| $B$ | $\cup$ | $E$ | $\vdash$ | $T$ |
|---|---|---|---|---|

```
mother(penelope,victoria).
mother(penelope,arthur).
father(christopher,victoria).
father(christopher,arthur).
```

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

```
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

# Induction of a classifier

- Given:
  - background knowledge $B$
  - a set of training examples $E$
  - a classification $c \in C$ for each example $e$
- Find: a theory $T$ (or *hypothesis*) such that $B \cup T \vdash c(e)$, for all $e \in E$

# Induction of a classifier: Train Example

- $B$: relations `has_car` and *car properties* (`length`, `roof`, `shape`, etc.)
  example.: `has_car(t1,c11)`, `shape(c11,bucket)`
- $E$: the trains `t1` to `t10`
- $C$: `east`, `west` (or $\neg$`east`)

- Possible $T$:

```
east(Train) :- has_car(Train,Car), length(Car,short), roof(Car,open).
```

# Learning as search

- Given:
  - Background knowledge $B$
  - Theory Description Language $T$ (logic)
  - Positives examples $P$ (class $\oplus$)
  - Negative examples $N$ (class $\ominus$)
  - A covering relation $covers(B, T, e)$
- Find: a theory that covers
  - all positive examples (completeness)
  - no negative examples (consistency)

# Learning as search

- Covering relation:

$$covers(B, T, e) \Leftrightarrow B \cup T \vdash e$$

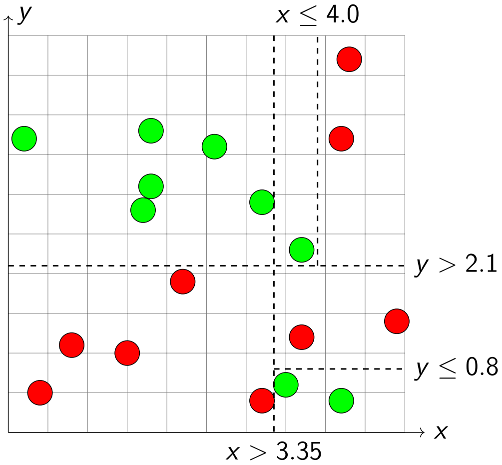
- A theory is a set of rules
  - a rule $R_i$ is in $CNF$ ( in the form: $l_1 \wedge l_2 \wedge \ldots l_n$ or $\{l_1, l_2, \ldots, l_n\}$)
  - the complete theory $T$ in $DNF$ (in the form $R_1 \vee R_2 \vee \ldots R_m$ or $\{R_1; R_2, \ldots; R_m\}$)
- Each rule is searched separately (efficiency)
- A rule must be consistent (cover no negatives), but not necessary complete
- Separate-and-conquer strategy
- Remove from P the examples already covered

# Separate-and-Conquer Strategy - Rule Learning Example

- Example (using propositional logic with just 2 numeric dimensions)

| Ex.# | X | Y | Class |
|---|---|---|---|
| 1 | 0.4 | 0.5 | $\ominus$ |
| 2 | 4.3 | 4.7 | $\ominus$ |
| 3 | 2.2 | 1.9 | $\ominus$ |
| 4 | 0.8 | 1.1 | $\ominus$ |
| 5 | 1.5 | 1.0 | $\ominus$ |
| 6 | 3.2 | 0.4 | $\ominus$ |
| 7 | 4.2 | 3.7 | $\ominus$ |
| 8 | 3.7 | 1.2 | $\ominus$ |
| 9 | 4.9 | 1.4 | $\ominus$ |
| 10 | 1.8 | 3.8 | $\oplus$ |
| 11 | 2.6 | 3.6 | $\oplus$ |
| 12 | 3.2 | 2.9 | $\oplus$ |
| 13 | 1.8 | 3.1 | $\oplus$ |
| 14 | 3.5 | 0.6 | $\oplus$ |
| 15 | 4.2 | 0.4 | $\oplus$ |
| 16 | 0.2 | 3.7 | $\oplus$ |
| 17 | 1.7 | 2.8 | $\oplus$ |
| 18 | 3.7 | 2.3 | $\oplus$ |

# Separate-and-Conquer Strategy - Rule Learning Example



# Separate-and-Conquer System - pFOIL

- Propositional variant of FOIL (Quinlan, 1993), one of the earliest and simplest relational learning system (*inductve logic programming* (ILP) - machine learning in predicate logic)
- Search heuristic: weighted information gain
- Search strategy: hill climbing
- Stopping criterion: encoding length restriction

# pFOIL - Algorithm

```
Pos ← positive examples;
Neg ← negative examples;
LearnedRules ← {};
while Pos ≠ {} do
    NewRule ← most general rule (⊤);
    NewRuleNeg ← Neg;
    while NewRuleNeg ≠ {} do
        NewRule ← argmax_{NewRule' ∈ ρ_s(NewRule)} pFOILGain(NewRule');
        NewRuleNeg ← cover(B, NewRule, NewRuleNeg);
        // NewRuleNeg = subset of NewRuleNeg satisfying NewRule
    end
    LearnedRules ← append(LearnedRules, (NewRule));
    Pos ← Pos \ cover(B, NewRule, Pos);
end
return LearnedRules
```

# pFOIL - Information Gain (pFOILGain)

$$pFOILGain(R \wedge L) = p_1 \left( log_2 \frac{p_1}{p_1+n_1} - log_2 \frac{p_0}{p_0+n_0} \right)$$

- where
  - $L$ = candidate literal to be added to rule $R$
  - $p_0$ = number of positive examples of $R$
  - $n_0$ = number of negative examples of $R$
  - $p_1$ = number of positive examples of $R \wedge L$
  - $n_1$ = number of negative examples of $R \wedge L$

# pFOIL - Stopping Criterion Based on Encoding Length Restriction

- Minimum Description Length (MDL) principle
- Training set of size $|T|$, a rule accounts for $p$ positive examples
- Number bits required to identify (choose) those examples
- Versus coding length per literal: $1 + log_2(|Literals|)$
- Tries to avoid learning complicated rules (covering only a few examples) ensuring that number of bits needed to encode a rule (clause) $<$ number of bits needed to encode the instances covered by it.

# Refinement Operator $\rho$ - Propositional Case

Rules are in conjunctive normal form (CNF):

- In the propositional case (examples are given in form of *attributes* and *values*), literals are just in the form *attribute* $=$ *value*.
- Example of $\rho$:
  - $R_0 = \top$
  - $R_1 \in \rho(R_0) = \{l_1, l_2, \ldots, l_n\}$, e.g. $R_1 = l_1$
  - $R_2 \in \rho(R_1) = \{l_1 \wedge l_2, l_1 \wedge l_3, \ldots, l_1 \wedge l_n\}$
  - $\ldots$
- i.e. $\rho(R)$ spezializes $R$ by adding a literal to the conjunction

# Refinement Operator $\rho$ - First-Order (FOIL) Case

Rules are in clausal normal form (CNF):

- The most general rule is the Horn clause using the predicate we want to learn, i.e.
  `east(T) :- .`
- $\rho(R)$ spezializes a rule $R$ by either:
  - adding $p(V_1, \ldots, V_n)$ where $p$ is a predicate and $V_i$ are variables not yet occurring in the rule(clause) $R$.
  - adding $equal(V_j, V_k)$ or its **negation**, where $V_j$ and $V_k$ are variables already present in the rule
- example:
  $\rho($`east(T):-.`$) = \{$`east(T):-has_car(A,B).`, `east(T):-roof(A,B).`$, \ldots\}$
- or $\rho($`east(T):-has_car(A,B).`$) =$
  $\{$`east(T):-has_car(A,B),roof(C,D).`,
  `east(T):-has_car(A,B),equal(T,A).`$, \ldots\}$

# Refinement Operator $\rho$ - General Overview

- Item set mining:
  - $\rho(I)$: add an item to current item set $I$
- Graph mining (data set of graph setting):
  - $\rho(G)$: add an edge to current sub graph $G$
- Learning rules (propositional):
  - $\rho(R_p)$: add a literal to current rule $R_p$
- Learning rules (first-order logic):
  - $\rho(R_f)$:
    - add a literal to current rule $R_f$ or
    - apply an elementary substitution to $R_f$.

# General Pattern Mining in Relation Data

- The idea of this refinement operator (and a bit more sophisticated ones) can also be applied to pattern mining.
- WARMR is one example of APRIORI in predite logic, others are CARMR, *etc*.
- However, the search space explodes if one is not careful:
  - use of so-called language bias
  - use of condensed representations
  - ...

# Outlook

- ProbLog
- relational learning and probailities