Relational Learning

CptS 570 – Machine Learning
School of EECS
Washington State University
Relational Learning

- Relational data
- Logic-based representation
- Graph-based representation
- Propositionalization
- Inductive Logic Programming (ILP)
- Graph-based relational learning
- Applications
Relational Data

- So far, training data have been propositional
  - Each instance represents one entity and its features

<table>
<thead>
<tr>
<th>ID</th>
<th>First Name</th>
<th>Last name</th>
<th>Age</th>
<th>Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>John</td>
<td>Doe</td>
<td>30</td>
<td>120,000</td>
</tr>
<tr>
<td>P2</td>
<td>Jane</td>
<td>Doe</td>
<td>29</td>
<td>140,000</td>
</tr>
<tr>
<td>P3</td>
<td>Robert</td>
<td>Smith</td>
<td>45</td>
<td>280,000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Learned hypotheses have also been propositional
  - If Income > 250,000 Then Rich
Relational Data

- Entities may be related to each other

<table>
<thead>
<tr>
<th>Married</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person1</td>
</tr>
<tr>
<td>P1</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

- Learned hypotheses should allow relations
  - If Income(Person1,Income1) and Income(Person2,Income2) and Married(Person1,Person2) and (Income1+Income2) > 250,000 Then RichCouple(Person1,Person2)
Representing Relational Data

- Logic-based representation
- Data
  - Person(ID, FirstName, LastName, Income)
    - Person(P1, John, Doe, 120,000)
  - Married(Person1, Person2)
    - Married(P1,P2), Married(P2,P1)
- Hypotheses
  - If Person(ID1, FirstName1, LastName1, Income1) and Person(ID2, FirstName2, LastName2, Income2) and Married(ID1,ID2) and (Income1 + Income2) > 250,000 Then RichCouple(ID1,ID2)
Representing Relational Data

- Graph-based representation
- Data

```
Person[Person]
  ID
  Age
  Income
  Married
Person[Person]
  ID
  Age
  Income
Person[Person]
  ID
  Age
  Income

<table>
<thead>
<tr>
<th>Last</th>
<th>First</th>
<th>ID</th>
<th>Age</th>
<th>Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doe</td>
<td>John</td>
<td>P1</td>
<td>30</td>
<td>120000</td>
</tr>
<tr>
<td>Doe</td>
<td>Jane</td>
<td>P2</td>
<td>29</td>
<td>140000</td>
</tr>
<tr>
<td>Smith</td>
<td>Robert</td>
<td></td>
<td>45</td>
<td>280000</td>
</tr>
</tbody>
</table>
```

CptS 570 – Machine Learning
Representing Relational Data

- Graph-based representation
- Hypotheses

```
Person X Income
Person Y Income
+ Operand Operand
\[
\text{Z} \quad 250000
\]
> Operand Operand
\[
\text{true}
\]
```
Using Relational Hypotheses

- Logical rule
  - Instance consists of relations
    - E.g., Person(), Married(), ...
  - Check if rule is matched by new instance
  - Unification (NP-Complete)

- Graphical rule
  - Instance consists of a graph
  - Check if rule matches a subgraph of instance
  - Subgraph isomorphism (NP-Complete)

- Many polynomial-time specializations exist (e.g., Horn clauses, trees)
Propositionalization

- Create new single table combining all relations

<table>
<thead>
<tr>
<th>First Name1</th>
<th>Last Name1</th>
<th>Age1</th>
<th>Income1</th>
<th>First Name2</th>
<th>Last Name2</th>
<th>Age2</th>
<th>Income2</th>
<th>Married</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Doe</td>
<td>30</td>
<td>120,000</td>
<td>Jane</td>
<td>Doe</td>
<td>29</td>
<td>140,000</td>
<td>Yes</td>
</tr>
<tr>
<td>Jane</td>
<td>Doe</td>
<td>29</td>
<td>140,000</td>
<td>Robert</td>
<td>Smith</td>
<td>45</td>
<td>280,000</td>
<td>No</td>
</tr>
<tr>
<td>Robert</td>
<td>Smith</td>
<td>45</td>
<td>280,000</td>
<td>Jane</td>
<td>Doe</td>
<td>29</td>
<td>140,000</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Apply propositional learner
- Number of fields in new table can grow exponentially
Terminology

- Relations are predicates (e.g., person, married)
- Predicate $p(a_1, a_2, \ldots, a_n)$ has arguments $a_1, a_2, \ldots, a_n$
- Arguments can be constants (e.g., sally) or variables (e.g., $X$, Income1)
- A predicate is ground if it contains no variables
Logic Programming

- Terminology
  - A literal is a predicate or its negation
  - A clause is a disjunction of literals
  - A Horn clause has at most one positive literal
  - A definite clause has exactly one positive literal
    \[(a \land b \rightarrow c) \equiv (\neg a \lor \neg b \lor c)\]

- Adopt Prolog syntax
  - Predicates and constants are lowercase
  - Variables are uppercase
  - E.g., married(X,Y), person(p1,john,doe,30,120000)
Inductive Logic Programming (ILP)

- **Given**
  - Training examples \((x^t, r^t) \in X\)
    - \(x^t\) is a set of facts (ground predicates)
    - \(r^t\) is a ground predicate
  - Background knowledge \(B\)
    - \(B\) is a set of predicates and rules (definite clauses)

- **Find hypothesis** \(h\) such that
  - \((\forall (x^t, r^t) \in X) \ B \land h \land x^t \vdash r^t\)
  - where \(\vdash\) means entails (can deduce)
Example
- Learn concept of child(X,Y): Y is the child of X
- r^t: child(bob,sharon)
- x^t: male(bob), female(sharon), father(sharon,bob)
- B: parent(U,V) ← father(U,V)
- h_1: child(X,Y) ← father(Y,X)
- h_2: child(X,Y) ← parent(Y,X)
FOIL [Quinlan, 1990]

- First-Order Induction of Logic (FOIL)
- Learns Horn clauses
- Set covering algorithm
  - Seeks rules covering subsets of positive examples
- Each new rule generalizes the learned hypothesis
- Each conjunct added to a rule specializes the rule
FOIL(Target Predicate, Predicates, Examples)

- \( Pos \) ← positive Examples
- \( Neg \) ← negative Examples
- while \( Pos \), do
  
  Learn a New Rule
  - \( NewRule \) ← most general rule possible
  - \( NewRuleNeg \) ← \( Neg \)
  - while \( NewRuleNeg \), do
    
    Add a new literal to specialize \( NewRule \)
    1. \( Candidate\_literals \) ← generate candidates
    2. \( Best\_literal \) ←
       \[ \arg\max_{L \in Candidate\_literals} \text{Foil\_Gain}(L, NewRule) \]
    3. add \( Best\_literal \) to \( NewRule \) preconditions
    4. \( NewRuleNeg \) ← subset of \( NewRuleNeg \) that satisfies \( NewRule \) preconditions
  
  - \( Learned\_rules \) ← \( Learned\_rules \) + \( NewRule \)
  - \( Pos \) ← \( Pos \) − \{members of \( Pos \) covered by \( NewRule \)\}
- Return \( Learned\_rules \)
Learning rule $p(X_1, X_2, \ldots, X_k) \leftarrow L_1 \land \ldots \land L_n$

Candidate specializations add new literal of form:
- $Q(V_1, \ldots, V_r)$ where at least one of $V_i$ must already exist as a variable in the rule
- $\text{Equal}(X_j, X_k)$ where $X_j$ and $X_k$ are variables present in the rule
- The negation of either of the above forms of literals
Information Gain in FOIL

\[
FoilGain(L, R) = t \left( \log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right)
\]

- \(L\) is the candidate literal to add to rule \(R\)
- \(p_0\) = number of positive bindings of \(R\)
- \(n_0\) = number of negative bindings of \(R\)
- \(p_1\) = number of positive bindings of \(R+L\)
- \(n_1\) = number of negative bindings of \(R+L\)
- \(t\) = number of positive bindings of \(R\) and \(R+L\)
- Note: \(-\log_2(p_0/(p_0 + n_0))\) = number of bits to indicate the class of a positive binding of \(R\)
FOIL Example

- **Target concept**
  - canReach(X,Y) true if directed path from X to Y

- **Examples**
  - Pairs of nodes for which path exists (e.g., <1,5>)
  - Graph described by literals
    - E.g., linkedTo(0,1), linkedTo(0,8)

- **Hypothesis space**
  - Horn clauses using predicates linkedTo and canReach
**FOIL Example: Input**

| X: 0, 1, 2, 3, 4, 5, 6, 7, 8. | 2,3  
| canreach(X,X)  | 0,1  
| 0,2  | 0,4  
| 0,5  | 0,6  
| 0,7  | 0,8  
| 1,2  | 1,4  
| 1,3  | 1,6  
| 1,5  | 1,7  
| 1,8  | 2,4  
| 2,5  | 2,6  
| 2,7  | 2,8  
| 3,4  | 3,5  
| 3,6  | 3,7  
| 3,8  | 4,5  
| 4,6  | 4,7  
| 4,8  | 5,6  
| 5,7  | 5,8  
| 6,7  | 6,8  
| 7,8  | ...  

| *linkedto(X,X) | 0,1  
| 0,2  | 1,2  
| 2,3  | 3,4  
| 3,8  | 4,5  
| 4,8  | 5,6  
| 6,7  | 6,8  
| 7,8  | ...  

CptS 570 – Machine Learning
FOIL Example: Output

<table>
<thead>
<tr>
<th>Relation canreach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation *linkedto</td>
</tr>
</tbody>
</table>

canreach:

State (36/81, 91.4 bits available)

- Save clause ending with linkedto(A,B) (cover 12, accuracy 100%)
- Save linkedto(C,B) (36,72 value 6.0)

Best literal linkedto(A,B) (4.6 bits)

Clause 0: canreach(A,B) :- linkedto(A,B).

...
FOIL Example: Output

State (24/69, 81.4 bits available)

Save clause ending with not(linkedto(C,A)) (cover 6, accuracy 85%)

Save linkedto(C,B) (24,60 value 4.8)
Save not(linkedto(B,A)) (24,57 value 6.5)

Best literal not(linkedto(C,A)) (4.6 bits)

State (6/7, 33.5 bits available)

Save clause ending with A<>B (cover 6, accuracy 100%)

Best literal A<>B (2.0 bits)

Clause 1: canreach(A,B) :- not(linkedto(C,A)), A<>B.

State (18/63, 71.5 bits available)

Save not(linkedto(B,A)) (18,51 value 5.4)

Best literal linkedto(C,B) (4.6 bits)

…
State (27/73 [18/54], 66.9 bits available)

Save clause ending with canreach(A,C) (cover 18, accuracy 100%)

Best literal canreach(A,C) (4.2 bits)

Clause 2: canreach(A,B) :- linkedto(C,B), canreach(A,C).

Delete clause
    canreach(A,B) :- not(linkedto(C,A)), A<>B.

    canreach(A,B) :- linkedto(A,B).
    canreach(A,B) :- linkedto(C,B), canreach(A,C).

Time 0.0 secs
ILP by Inverse Resolution

- Resolution rule
  \[ \frac{P \lor L}{\neg L \lor R} \]
  \[ \frac{\neg L \lor R}{P \lor R} \]

- Given initial clauses \( C_1 \) and \( C_2 \), find a literal \( L \) from clause \( C_1 \) such that \( \neg L \) occurs in clause \( C_2 \).

- Form the resolvent \( C \) by including all literals from \( C_1 \) and \( C_2 \), except for \( L \) and \( \neg L \):
  - \( C = (C_1 - \{L\}) \cup (C_2 - \{\neg L\}) \)
  - where \( \cup \) denotes set union, and “\(-\)” denotes set difference.
Inverting Resolution

\[
\begin{align*}
C_1 & : \text{PassExam} \lor \neg \text{KnowMaterial} \\
C_2 & : \text{KnowMaterial} \lor \neg \text{Study} \\
C & : \text{PassExam} \lor \neg \text{Study}
\end{align*}
\]
Inverting Resolution

- Propositional
- Given initial clauses $C_1$ and $C$, find a literal $L$ that occurs in clause $C_1$, but not in clause $C$
- Form the second clause $C_2$ by including the following literals
  - $C_2 = (C - (C_1 - \{L\})) \cup \{\neg L\}$
Inverting Resolution

- **First-order resolution**
  - Find a literal $L_1$ from clause $C_1$, literal $L_2$ from clause $C_2$, and substitution $\theta$ such that $L_1\theta = \neg L_2\theta$
  - Form the resolvent $C$ by including all literals from $C_1\theta$ and $C_2\theta$, except for $L_1\theta$ and $\neg L_2\theta$
  - $C = (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta$

- **Inverting first-order resolution**
  - $C_2 = (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{-L_1\theta_1\theta_2^{-1}\}$
Inverting Resolution

\[
\begin{align*}
\text{Father (Tom, Bob)} & \quad \Rightarrow \quad \text{GrandChild}(y, x) \lor \neg \text{Father}(x, z) \lor \neg \text{Father}(z, y) \\
\{\text{Bob}/y, \text{Tom}/z\} & \quad \Rightarrow \quad \{\text{Bob}/y, \text{Tom}/z\} \\
\text{Father (Shannon, Tom)} & \quad \Rightarrow \quad \text{GrandChild}(Bob, x) \lor \neg \text{Father}(x, Tom) \\
\{\text{Shannon}/x\} & \quad \Rightarrow \quad \{\text{Shannon}/x\} \\
\text{GrandChild}(Bob, Shannon) & \quad \Rightarrow \quad \text{GrandChild}(Bob, Shannon)
\end{align*}
\]
Reduce combinatorial explosion by generating the most specific acceptable $h$

User specifies $H$ by stating predicates, functions, and forms of arguments allowed for each

Progol uses set covering algorithm
- For each $<x^t, r^t>$
- Find most specific hypothesis $h_t$ s.t. $B \land h_t \land x^t \vdash r^t$
  - actually, considers only $k$-step entailment

Conduct general-to-specific search bounded by specific hypothesis $h_t$, choosing hypothesis with minimum description length
% Learning aunt_of from parent_of and sister_of.

% Settings

:- set(posonly)?

% Mode declarations

:- modeh(1,aunt_of(+person,+person))?  
:- modeb(*,parent_of(+person,+person))?  
:- modeb(*,parent_of(-person,+person))?  
:- modeb(*,parent_of(+person,-person))?  
:- modeb(*,sister_of(+person,-person))?  

% Types

person(jane).
person(henry).
person(sally).
person(jim).
person(sam).
person(sarah).
person(judy).
...

Prolog Example: Input

% Background knowledge


father_of(sam,henry).
mother_of(sarah,jim).
sister_of(jane,sam).
sister_of(sally,sarah).
sister_of(judy,sarah).

% Examples

aunt_of(jane,henry).
aunt_of(sally,jim).
aunt_of(judy,jim).
Prolog Example: Output

CProlog Version 4.4

[Noise has been set to 100%]
[Example inflation has been set to 400%]
[The posonly flag has been turned ON]
[\:- set(posonly)? – Time taken 0.00s]
[\:- modeh(1,aunt_of(+person,+person))? – Time taken 0.00s]
[\:- modeb(100,parent_of(-person,+person))? – Time taken 0.00s]
[\:- modeb(100,parent_of(+person,-person))? – Time taken 0.00s]
[\:- modeb(100,sister_of(+person,-person))? – Time taken 0.00s]
[Testing for contradictions]
[No contradictions found]
[Generalising aunt_of(jane,henry).]
[Most specific clause is]

aunt_of(A,B) :- parent_of(C,B), sister_of(A,C).
...

Prolog Example: Output

[Learning aunt_of/2 from positive examples]
[C:-0,12,11,0 aunt_of(A,B).]
[C:6,12,4,0 aunt_of(A,B) :- parent_of(C,B).]
[C:6,12,3,0 aunt_of(A,B) :- parent_of(C,B), sister_of(A,C).]
[C:6,12,3,0 aunt_of(A,B) :- parent_of(C,B), sister_of(A,D).]
[C:4,12,6,0 aunt_of(A,B) :- sister_of(A,C).]
[5 explored search nodes]
f=6,p=12,n=3,h=0
[Result of search is]

aunt_of(A,B) :- parent_of(C,B), sister_of(A,C).

[3 redundant clauses retracted]
aunt_of(A,B) :- parent_of(C,B), sister_of(A,C).

[Total number of clauses = 1]

[Time taken 0.00s]
Relational Decision Trees

- TILDE [Blockheel & De Raedt, 1998]
  - Relational extension to C4.5
RIBL [Emde and Wettschereck, 1996]

Distance measure compares top-level objects, then objects related to them, and so on

```
member(person1, 45, male, 20, gold)
member(person2, 30, female, 10, platinum)

car(person1, wagon, 200, volkswagen)
car(person1, sedan, 220, mercedesbenz)
car(person2, roadster, 240, audi)
car(person2, coupe, 260, bmw)

house(person1, mурgle, 1987, 560)
house(person1, montecarlo, 1990, 210)
house(person2, mурgle, 1999, 430)

district(montecarlo, famous, large, monaco)
district(mурgle, famous, small, slovenia)
```
Graph-based Relational Learning

- Unsupervised learning
  - Frequent subgraphs
  - Compressing subgraphs

- Supervised learning
  - Graph features
  - Graph kernels
  - Relational decision trees
  - Relational instance-based learning
**Terminology**

- **Graph** $G = (V,E)$, where $V$ is a set of **vertices** and $E$ is a set of **edges** $(u,v)$ such that $u,v \in V$
- Edges may be **directed** or **undirected**
- Vertices and/or edges may have **labels**
- Graph $G_1=(V_1,E_1)$ is **isomorphic** to $G_2=(V_2,E_2)$ if there exists a bijective mapping $f: V_1 \rightarrow V_2$ such that $(u,v) \in E_1$ iff $(f(u),f(v)) \in E_2$
  - The complexity of checking if two graphs are isomorphic is unknown, but expensive
**Terminology**

- Graph $G_1=(V_1,E_1)$ is a **subgraph** of $G_2=(V_2,E_2)$ if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$.

- $G_1$ is an **induced subgraph** of $G_2$ if $V_1 \subseteq V_2$ and $E_1 = \{(u,v) \in E_2 \mid u, v \in V_1\}$.

- **Subgraph isomorphism**: $G_1$ is isomorphic to a subgraph of $G_2$.
  - Subgraph isomorphism is NP–Complete.
Graph Invariants

- **Adjacency matrix**

\[
\begin{pmatrix}
  v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\
  0 & \{e_2, e_3, e_4\} & 0 & \{e_1\} & 0 & 0 \\
  \{e_2, e_3, e_4\} & 0 & \{e_5\} & 0 & 0 & 0 \\
  0 & \{e_5\} & 0 & \{e_6\} & 0 & 0 \\
  \{e_1\} & 0 & \{e_6\} & \{e_7\} & \{e_8, e_9\} & 0 \\
  0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

- All permutations of rows and columns yield an identical (isomorphisms) graph

- **Canonical form**
  - Generate code by concatenating rows of adjacency matrix
  - Choose code that is lexicographically smallest or largest
Graph Invariants

- Minimum DFS code
  - Concatenate edges of DFS trees
  - Take smallest lexicographic order $\min(G)$
  - Canonical label
  - Two graphs $G_1$ and $G_2$ isomorphic iff $\min(G_1) = \min(G_2)$

![Depth-First Search Tree Diagram]

**Figure 1. Depth-First Search Tree**

<table>
<thead>
<tr>
<th>edge</th>
<th>(Fig 1b) $\alpha$</th>
<th>(Fig 1c) $\beta$</th>
<th>(Fig 1d) $\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$(0, 1, X, a, Y)$</td>
<td>$(0, 1, Y, a, X)$</td>
<td>$(0, 1, X, a, X)$</td>
</tr>
<tr>
<td>1</td>
<td>$(1, 2, Y, b, X)$</td>
<td>$(1, 2, X, a, X)$</td>
<td>$(1, 2, X, a, Y)$</td>
</tr>
<tr>
<td>2</td>
<td>$(2, 0, X, a, X)$</td>
<td>$(2, 0, X, b, Y)$</td>
<td>$(2, 0, Y, b, X)$</td>
</tr>
<tr>
<td>3</td>
<td>$(2, 3, X, c, Z)$</td>
<td>$(2, 3, X, c, Z)$</td>
<td>$(2, 3, Y, b, X)$</td>
</tr>
<tr>
<td>4</td>
<td>$(3, 1, Z, b, Y)$</td>
<td>$(3, 0, Z, b, Y)$</td>
<td>$(3, 0, Z, c, X)$</td>
</tr>
<tr>
<td>5</td>
<td>$(1, 4, Y, d, Z)$</td>
<td>$(0, 4, Y, d, Z)$</td>
<td>$(2, 4, Y, d, Z)$</td>
</tr>
</tbody>
</table>

**Table 1. DFS codes for Fig. 1(b)-(d)**
Frequent Subgraph Mining

- Given a set of graphs $G$ and minimum support threshold $t$
- Find all subgraphs $g_s$ such that

$$\frac{|\{g \in G \mid g_s \subseteq g\}|}{|G|} \geq t$$

- where $g_s \subseteq g$ mean $g_s$ is isomorphic to a subgraph of $g$
Frequent Subgraph Mining

- **Systems**
  - Frequent Sub-Graph discovery (FSG)
    - Kuramochi and Karypis, 2001
  - Graph-based Substructure pattern mining (gSpan)
    - Yan and Han, 2002 (uses DFS codes)
  - Apriori-based Graph Mining (AGM)
    - Inokuchi, Washio and Motoda, 2003
  - GrAph/Sequence/Tree extractiON (GASTON)
    - Nijssen and Kok, 2004 ([www.liacs.nl/~snijssen/gaston](http://www.liacs.nl/~snijssen/gaston))

- Focus on pruning and fast, code-based graph matching
Compressing Subgraphs

- Minimum Description Length (MDL) principle
  - Best theory minimizes description length of theory and the data given theory
- Best subgraph $S$ minimizes description length of subgraph definition $DL(S)$ and compressed graph $DL(G/S)$

$$\min_S (DL(S) + DL(G | S))$$
Graph Compression
Compressing Subgraphs

- Systems
  - Graph-Based Induction (GBI)
    - Yoshida, Motoda and Indurkhya, 1994
  - SUBstructure Discovery Using Examples (SUBDUE)
    - Cook and Holder, 1994
- Focus on efficient subgraph generation and compression-based heuristic search
SUBDUE: Sample Input

sample.g:

v 1 object e 1 11 shape
v 2 object e 2 12 shape
v 3 object e 3 13 shape
v 4 object e 4 14 shape
e 5 object e 5 15 shape
e 6 object e 6 16 shape
e 7 object e 7 17 shape
e 8 object e 8 18 shape
e 9 object e 9 19 shape
e 10 object e 10 20 shape
e 11 triangle e 1 5 on
e 12 triangle e 2 6 on
e 13 triangle e 3 7 on
e 14 triangle e 4 8 on
v 15 square e 5 10 on
v 16 square e 9 10 on
v 17 square e 10 2 on
v 18 square e 10 3 on
v 19 circle e 10 4 on
e 20 rectangle

CptS 570 – Machine Learning
### SUBDUE 5.2.1

**Parameters:**
- Input file: sample.g
- Predefined substructure file: none
- Output file: none
- Beam width: 4
- Compress: false
- Evaluation method: MDL
- 'e' edges directed: true
- Incremental: false
- Iterations: 1
- Limit: 9
- Minimum size of substructures: 1
- Maximum size of substructures: 20
- Number of best substructures: 3
- Output level: 2
- Allow overlapping instances: false
- Prune: false
- Threshold: 0.000000
- Value-based queue: false
- Recursion: false

...
SUBDUE: Sample Output

Read 1 total positive graphs

1 positive graphs: 20 vertices, 19 edges, 252 bits
7 unique labels

3 initial substructures

Best 3 substructures:

(1) Substructure: value = 1.86819, pos instances = 4, neg instances = 0
    Graph(4v,3e):
        v 1 object
        v 2 object
        v 3 triangle
        v 4 square
        d 1 3 shape
        d 2 4 shape
        d 1 2 on

...
(2) Substructure: value = 1.37785, pos instances = 4, neg instances = 0
Graph(3v,2e):
  v 1 object
  v 2 object
  v 3 square
  d 2 3 shape
  d 1 2 on

(3) Substructure: value = 1.37219, pos instances = 4, neg instances = 0
Graph(3v,2e):
  v 1 object
  v 2 object
  v 3 triangle
  d 1 3 shape
  d 1 2 on

SUBDUE done (elapsed CPU time = 0.00 seconds).
Given positive graphs $G^+$ and negative graphs $G^-$

Find subgraph $S$ minimizing $DL(G^+|S) / DL(G^-|S)$

If $|G^+| \gg 1$ and $|G^-| \gg 1$, find subgraph $S$ minimizing

$$error = \left| \{S \not\subset G \mid G \in G^+\} \right| + \left| \{S \subset G \mid G \in G^-\} \right|$$

$$\left| G^+ \right| + \left| G^- \right|$$

Positive Graphs  

Negative Graphs  

SUBDUE  

Pattern(s)
Supervised Learning

- DT-GBI (Geamsakul et al., 2003)
  - Decision Tree Graph-based Induction
- Graph instance-based learning
  - Graph edit distance
    - Minimum cost changes to transform one graph into another (add/delete/change node, edge, direction, label)
Use feature vector based on presence of frequent or compressing subgraphs

Graph kernels
  - Compare substructures of graphs
    - E.g., walks, paths, cycles, trees, subgraphs
  - \( K(G_1, G_2) = \) number of identical random walks in both graphs
  - \( K(G_1, G_2) = \) number of subgraphs shared by both graphs
ILP Applications

- Natural language understanding
- Drug design
- Protein structure prediction (→)

Figure 14.2: Rules for predicting α-helix secondary structure.

[Muggleton et al., 1992]
Graph-based Applications

- Social networks (→)
- Computer networks
- WWW, Internet

- Biological networks (→)
- Drug design
Summary: Relational Learning

- Exploit relational information in data
- Inductive Logic Programming (ILP)
  - FOIL, Progol
- Graph-based relational learning
  - SUBDUE
- Numerous applications
- Computationally expensive