Road Map

- Relational Classifiers
- Collective Classification
- Advanced SRL Models
  - Background: Graphical Models
  - Key Ideas: Par-factor graphs
  - Languages
Road Map

- Relational Classifiers
- Collective Classification
- Advanced SRL Models
  - Background: Graphical Models
  - Key Ideas: Par-factor graphs
  - Languages
Probabilistic Queries

- Given a set of random variables:
  - For simplicity, assume Boolean-valued
    - True
    - False
    - Unknown
  - e.g., describing one’s travel experience
    - Rainy?
    - Foggy?
    - Windy?
    - FlightOnTime?

- For simplicity, assume Boolean-valued
  - In general, they are not independent!

- Want to be able to answer questions about their values
Probabilistic Queries: Marginals

- What is the marginal probability of one of the variables, e.g.,

\[ P(\text{FlightOnTime}) = ? \]

- Often, we want to condition on observed variables

\[ P(\text{FlightOnTime} | \text{Foggy}, \text{Windy}) = ? \]
Probabilistic Queries: Most Likely

- What is the most likely joint assignment of values to the variables

  e.g., what is the most likely configuration of weather conditions and flight status?

- In general, not the same as setting each variable to its most likely value!
**Probabilistic Queries**

- Such queries are easy to answer if we know the joint probability distribution:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>0.02</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.11</td>
</tr>
</tbody>
</table>

Problem: Number of parameters grows exponentially in number of variables

© Getoor & Mihalkova 2010-2011
Naïve Solution

Assume the variables are independent:

<table>
<thead>
<tr>
<th></th>
<th>0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td></td>
</tr>
</tbody>
</table>

Now the number of parameters is linear in number of variables, but the assumption is typically false!
Two Extremes

The true distribution is somewhere in the middle: some (conditional) independencies

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>0.02</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.09</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
</tr>
</tbody>
</table>

2^n parameters, no assumptions

n parameters, strong independence assumption

© Getoor & Mihalkova 2010-2011
Explicitly model the conditional independencies among the variables, thus representing joint probability distributions over large sets of variables compactly. 

Probabilistic Graphical Models
Conditional Independence

- A variable V’s neighbors = Set of variables that are connected to the same variables as V.
- V’s neighbors render it conditionally independent from all other variables.

Rainy?  Windy?  Foggy?

FlightOnTime?

rendered independent given neighbors’ values

© Getoor & Mihalkova 2010-2011
Factor Graphs

- Bipartite graph containing two kinds of nodes
  - Variables $y_1$
  - Factors $\bullet$, $\blacksquare$: strictly positive functions of the variables to which they are connected in the graph
The probability of a joint assignment of values \( y \) to the set of variables \( Y \) is computed as:

\[
P(Y = y) = \frac{\prod_{f \in \text{Factors}} f(y_{\{f\}})}{Z}
\]
Factor Graphs

- What does the no-independencies factor graph look like?
Factor Graphs

What does the no-dependencies factor graph look like?
Factor Graphs: Log-Linear Rep

- Each orange square represents \( \exp(\theta_L \cdot f_i(y_i)) \)
- Each black square represents \( \exp(\theta_G \cdot f_{i,j}(y_i, y_j)) \)

© Getoor & Mihalkova 2010-2011
More Generally…

- Factors can be functions of any number of variables

  However, to keep the model compact, we want to keep factors small. In the worst case, the number of parameters needed by a factor is exponential in the number of variables of which it is a function.

- Not all pairs of variables have to share a factor

  In fact, we want to avoid having variables share factors unless there truly is a dependence between them.

- Factors can be computed by any function that returns a strictly positive value

  The log-linear representation is convenient and has nice properties.
Markov Nets

- Markov networks (aka Markov random fields) can be viewed as special cases of factor graphs:

Same Markov net could indicate that $y_2$, $y_3$, and $y_4$ share a single factor.

Equivalent expressivity. However, factor graphs are more explicit.
Markov Nets Continued

- Factors are called potential functions
- Viewed as functions that ensure compatibility between assignments to the nodes

For example, in the Ising Model the possible assignments are \{-1, +1\}, and one has:

\[ \phi_{i,j} = \exp(\theta_{i,j} y_i y_j) \]

Positive, or ferromagnetic, \(\theta_{i,j}\) encourages neighboring nodes to have the same assignment.

Negative, or anti-ferromagnetic, \(\theta_{i,j}\) encourages contrasting assignment.

Variables participating in shared potential functions form cliques in the graph.
Markov networks

The probability of a joint assignment of values \( y \) to the set of variables \( Y \) is computed as:

\[
P(Y = y) = \frac{\prod_{C \in \text{Clique}} \phi(y_C)}{Z}
\]

- Normalizing constant
- Variables in clique \( C \)
Markov networks

- ... assuming the log-linear representation for the clique potentials:

$\phi(y_C) = \exp(\theta_C f(y_C))$

- The joint probability becomes:

$P(Y = y) = \frac{\exp(\sum_{C \in \text{Cliques}} \theta_C f(y_C))}{Z}$
Markov Nets: Transitivity

- How to encode transitivity?

  **Want to say:** If $A$ is friends with $B$ and $B$ is friends with $C$, then $A$ is friends with $C$. For all permutations of the letters.

- Model as a Markov net with a node for each decision, connecting dependent decisions in cliques

- Possible assignments: 1 (friends), 0 (not friends)
Quick Aside: Two Kinds of Graphs

We often draw social networks like this:

```
A ─── Friends ─── B
   |       |       |
   V       V       V
C ─── Friends ─── C
```

**Relational Graph:**
- Nodes represent entities
- Edges represent relationships

... not to be confused with a Markov net:

```
γ_1 = (A<->B) ─── γ_2 = (B<->C) ─── γ_3 = (A<->C)
```

**Markov Net:**
- Nodes represent decisions
- Edges represent dependencies between decisions

© Getoor & Mihalkova 2010-2011
Quick Aside: Two Kinds of Graphs

In Part I we were drawing social networks like this:

**Relational Graph:**
- Nodes represent entities
- Edges represent relationships

\[ y_3 = (A \leftrightarrow C) \]

... not to be confused with a Markov net:

**Markov Net:**
- Nodes represent decisions
- Edges represent dependencies between decisions

\[ y_1 = (A \leftrightarrow B) \]
\[ y_2 = (B \leftrightarrow C) \]
\[ y_3 = (A \leftrightarrow C) \]

Since here we are trying to infer the presence of a relationship, our Markov Net has a node for each possible edge in the Relational graph.

© Getoor & Mihalkova 2010-2011
Markov Nets: Transitivity

- How to encode transitivity?

  Want to say: If $A$ is friends with $B$ and $B$ is friends with $C$, then $A$ is friends with $C$. For all permutations of the letters.

- Model as a Markov net with a node for each decision, connecting dependent decisions in cliques

- Possible assignments: 1 (friends), 0 (not friends)
Markov Nets: Transitivity

<table>
<thead>
<tr>
<th>$\gamma_1 = (A \leftrightarrow B)$</th>
<th>$\gamma_2 = (B \leftrightarrow C)$</th>
<th>$\gamma_3 = (A \leftrightarrow C)$</th>
<th>$\phi_{1,2,3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\sim e^0$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$\sim e^0$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$\sim e^0$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$x e^{-\theta}$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$x e^{-\theta}$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$x e^{-\theta}$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$\checkmark e^\theta$</td>
</tr>
</tbody>
</table>

... one possibility

© Getoor & Mihalkova 2010-2011
If A and B are enemies and B and C are enemies, then A and C are friends. For all permutations of the letters.

<table>
<thead>
<tr>
<th>$y_1 = (A \leftrightarrow B)$</th>
<th>$y_2 = (B \leftrightarrow C)$</th>
<th>$y_3 = (A \leftrightarrow C)$</th>
<th>$\phi_{1,2,3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\times e^{-\theta}$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$e^\theta$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$e^\theta$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$e^0$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$e^\theta$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$e^0$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$e^0$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$e^0$</td>
</tr>
</tbody>
</table>

© Getoor & Mihalkova 2010-2011
To cast a Bayesian net as a factor graph, include a factor as a function of each node and its parents.

Going the other way requires ensuring acyclicity.
To cast a Bayesian net as a factor graph, include a factor as a function of each node and its parents.

Here the factors take the shape of conditional probability tables, giving, for each configuration of assignments to the parents, the distribution over assignments to the child.

Automatically normalized!
Bayesian Nets

- The probability of a joint assignment of values $\mathbf{y}$ to the set of variables $\mathbf{Y}$ is computed as:

$$P(\mathbf{Y} = \mathbf{y}) = \prod_{\mathbf{y}_i \in \text{Nodes}} P(y_i | \text{Pa}(y_i))$$

Parents of $y_i$
Inference in Graphical Models

- Two flavors:
  - Computing marginal probabilities of the variables
  - Finding the most likely joint assignment to the variables

- Variety of algorithms exist:
  - Variable elimination
  - Message passing, e.g., belief propagation
  - Sampling, e.g., Gibbs sampling
  - (Integer) linear programming

- Next we show a quick overview of Gibbs sampling
Gibbs Sampling

- Assign initial values to the variables
- While there is time:
  - For each variable i:
    - Sample a new value for i
    - Factor values are computed using current assignments to other participating variables
Gibbs Sampling

\[
x \in \{0, 1\}
\]

\[
P(i = x) = \frac{\phi_i(x)\phi_{i,j,k}(x, 0, 1)\phi_{i,m}(x, 1)\phi_{i,n}(x, 0)}{\phi_i(0)\phi_{i,j,k}(0, 0, 1)\phi_{i,m}(0, 1)\phi_{i,n}(0, 0) + \phi_i(1)\phi_{i,j,k}(1, 0, 1)\phi_{i,m}(1, 1)\phi_{i,n}(1, 0)}
\]

Normalize by summing over every possible value of \(x\)
Gibbs Sampling

- In general...

  \[ \mathcal{A} \] : Current assignment of values to all variables in the model

  \[ \mathcal{A}[V] \] : Assignment of values to all variables in set \( V \)

  \( \Phi_i \) : Set of factors in which variable \( i \) participates

  \( V_\phi \) : Set of variables used by factor \( \phi \)

  \( \chi \) : Set of possible values

\[
P(i = x) = \frac{\prod_{\phi \in \Phi_i} \phi(i = x, V_\phi \setminus \{i\} = \mathcal{A}[V_\phi \setminus \{i\}])}{\sum_{x' \in \chi} \prod_{\phi \in \Phi_i} \phi(i = x', V_\phi \setminus \{i\} = \mathcal{A}[V_\phi \setminus \{i\}])}
\]

Set other vars in the factor to their current value
Gibbs Sampling vs ICA

- Algorithmically, Gibbs sampling and ICA are very similar
- There are several important distinctions

<table>
<thead>
<tr>
<th>Criterion</th>
<th>ICA</th>
<th>Gibbs Sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>End Goal</td>
<td>Find most likely joint assignment to vars</td>
<td>Compute posterior marginal probabilities for vars</td>
</tr>
<tr>
<td>In each iteration…</td>
<td>Set the most probable value based on current assignments</td>
<td>Sample a new value based on current assignments</td>
</tr>
<tr>
<td>Graph over which it is performed</td>
<td>Relational graph</td>
<td>Factor graph</td>
</tr>
<tr>
<td>Initialization</td>
<td>Bootstrap from local features</td>
<td>Random/MAP state + burn-in</td>
</tr>
</tbody>
</table>

© Getoor & Mihalkova 2010-2011
Learning in Graphical Models

- Learning the parameters
  - In a Markov network with a log-linear representation of the potential functions:
    \[ \exp(\theta f(y_f)) \]
  - In a Bayesian network, the graph is given, want to learn conditional probability distributions for each child given parents

- Learning the structure
Parameter Learning

- Given: A data set $\mathcal{D}$, and a factor graph $\mathcal{F}$, whose factors are parameterized by a vector $\theta$
- Goal: Estimate values for $\theta$ to maximize prob. of the data

$$\theta^* = \arg\max_{\theta} P(\mathcal{D}|\mathcal{F}_\theta)$$

- This time, it will actually be helpful to consider Markov nets and Bayesian nets separately

This is one possible criterion for optimizing weights. More exist, e.g. Bayesian learning
… in Bayesian nets

- Easy! (From fully observed data)
- Just collect empirical counts

\[
P(y_1 = x | y_2 = \alpha, y_4 = \beta) = \frac{\text{count}(y_1 = x, y_2 = \alpha, y_4 = \beta)}{\sum_{x'} \text{count}(y_1 = x', y_2 = \alpha, y_4 = \beta)}
\]

- Use EM if there are missing values
… in Markov nets

- Cannot be computed in closed form
- Use gradient descent or any other optimization procedure
- If we use a log-linear model with one parameter per factor,
  
  gradient wrt $\theta_i$ corresponding to factor $\phi_i$ is given by:

$$\frac{\partial}{\partial \theta_i} = \phi_i(D) - \mathbb{E}_{\theta_i}(\phi_i)$$

Value in the data

Expected value according to current estimate
Road Map

- Relational Classifiers
- Collective Classification
- Advanced SRL Models
  - Background: Graphical Models
  - Key Ideas: Par-factor graphs
  - Languages
Par-factor Graphs

- Factor graphs with parameterized factors
  - Terminology introduced by [Poole, IJCAI03]
- A par-factor is defined as the triple
  - $\mathcal{A}$: set of parameterized random variables
  - $f$: function that operates on these variables and evaluates to $> 0$
  - $\mathcal{C}$: set of constraints
- A par-factor graph is a set of par-factors
Parameterized Random Vars

- Blueprint for manufacturing random variables
- For example:
  - Let $A$ and $B$ be variables, then $A \leftrightarrow B$ is a parameterized random variable.
  - Given specific individuals (Ann, Bob, Don, …, Xin, Yan), we can manufacture random variables from it by replacing $A$ and $B$ with the given individuals in all possible ways.
Parameterized Random Vars

- Can be viewed as a blueprint for manufacturing random variables
- For example:
  - Let A and B be variables, then is a parameterized random variable.
  - Given specific individuals, we can manufacture random variables: Ann<->Bob, Ann<->Don, Xin<->Yan...

We call this instantiating the parameterized RV.
The constraints in set $C$ govern how par-RVs can be instantiated.

For example, one constraint for our par-RV could be that $B \neq Don$.

With this constraint, the possible instantiations are:

- Ann$\leftrightarrow$Bob
- Ann$\leftrightarrow$Don
- Xin$\leftrightarrow$Yan
- ...
Transitivity Par-factor

$\mathcal{A} = \{ A \leftrightarrow B, B \leftrightarrow C, A \leftrightarrow C \}$

$f$ can be defined as before

$C = \{ A \neq B \neq C \}$

However, whereas before these referred to the potential friendships of specific individuals, now they refer to variables, i.e. to people in general.
**Transitivity Par-factor**

\[ \mathcal{A} = \{ A \leftrightarrow B, B \leftrightarrow C, A \leftrightarrow C \} \]

- \( f \) can be defined as before

- \( C = \{ A \neq B \neq C \} \)

However, whereas before these referred to the potential friendships of **specific individuals**, now they refer to **variables**, i.e. to people in general.

This means that now we can train on one set of individuals and apply our models to an entirely different set.
To instantiate a par-factor, we need a set of individuals: Ann, Bob, Don.

Then we consider all possible instantiations of the par-RVs with these individuals:

- Ann<->Bob
- Ann<->Don
- Bob<->Don
- Bob<->Ann
- Don<->Bob
- Don<->Ann

... etc.
To instantiate a par-factor, we need a set of individuals. Then we consider all possible instantiations of the par RVs with these individuals:

- Ann <-> Bob
- Ann <-> Don
- Bob <-> Don
- Bob <-> Ann
- Don <-> Bob
- Don <-> Ann

... etc.

**Moral of the story:**
So much power can be dangerous!

- Starting with just 3 individuals, we've ended up with a huge and densely connected graph.
- Inference becomes very problematic.
Managing our Power

- **Constraints**
  - One way of keeping the factor graph size manageable is by imposing appropriate constraints on permitted instantiations

- **Par-factor size**
  - More par-RVs per par-factor translate into more RVs per factor

- When defining a par-factor, it is important to think:
  - How many instantiations will this par-factor have?
  - How many RVs per instantiation?

- This is easier said than done
  - Will discuss more

© Getoor & Mihalkova 2010-2011
To instantiate a par-factor, we need a set of individuals: Ann, Bob, Don.

Then we consider all possible instantiations of the par-RVs with these individuals:
Another Example

\[ A = \begin{cases} \text{Friends (A, B), A and B are people,} \\ \text{F1 and F2 are flavors} \\ \end{cases} \]

\[ C = \begin{cases} \text{Ann, Bob, Eve} \\ \text{Vanilla, Cherry} \\ \text{Friends(Ann, Bob)} \\ \text{Friends(Bob, Eve)} \\ \end{cases} \]

\[ f = \begin{cases} \exp(\theta) & \text{If } F1 = F2 \& \text{Otherwise} \\ 1 & \end{cases} \]
Parameter Tying

- Factors with tied parameters
  - Means that they share their parameter vectors
  - Can view them as a function that gets evaluated for different (sets of) nodes in the graph

- Advantages of tying:
  - Fewer parameters to estimate
    - Avoid overfitting
    - More robust estimation
  - Better generalization
    - E.g., we learn about transitivity in general, not about the transitivity between Ann, Bob, and Carl’s friendships

- Parameter learning can be easily extended to learn with tied parameters
Recap So Far

- Extended factor graphs to allow for convenient parameter tying
  - Parameter learning: an extension of parameter learning in Bayesian/Markov nets
  - Inference: instantiate the par-factors and perform inference as before

- Are we done?
  - We still do not have a convenient language for specifying the function part of a par-factor
  - A wide range of languages have been introduced and studied in the field of statistical relational learning (SRL). Here we review just a few
SRL Road Map

Factor Graphs

Bayesian Nets

Markov Nets

Par-factor Graphs

Directed Models

BLPs [Kersting & De Raedt, ILP01]
PRMs [Koller & Pfeffer, AAAI98]
etc.

Undirected Models

RMNs [Taskar et al., UAI02]
MLNs [Richardson & Domingos, MLJ06]
etc.

Hybrid Models

RDNs [Neville & Jensen, JMLR07]

© Getoor & Mihalkova 2010-2011
Directed Models

- Bayesian logic programs (BLPs)
  - Based on first-order logic
  - [Kersting & De Raedt, ILP01]

- Probabilistic relational models (PRMs)
  - Using an object-oriented, frame-based representation
  - [Koller & Pfeffer, AAAI98]
Describes the types of objects and relations in the database
Probabilistic Relational Model

Author
- Smart
- Good Writer

Paper
- Quality
- Accepted

Review
- Mood
- Length

© Getoor & Mihalkova 2010-2011
Probabilistic Relational Model

\[
P(\text{Paper.Accepted} | \text{Paper.Quality}, \text{Paper.Review.Mood})
\]

© Getoor & Mihalkova 2010-2011
Probabilistic Relational Model

Author

Good Writer

Smart

Review

Mood

Length

Paper

Quality

Accepted

| Q, M | P(A | Q, M) |
|------|------------|
| f, f | 0.1 0.9    |
| f, t | 0.2 0.8    |
| t, f | 0.6 0.4    |
| t, t | 0.7 0.3    |

© Getoor & Mihalkova 2010-2011
Fixed relational skeleton $\sigma$:
- set of objects in each class
- relations between them
PRM defines distribution over instantiations of attributes
A Portion of the BN

| Q, M | P(A | Q, M) |
|------|-----------|
| f, f | 0.1 0.9   |
| f, t | 0.2 0.8   |
| t, f | 0.6 0.4   |
| t, t | 0.7 0.3   |
A Portion of the BN

\[ P(A | Q, M) \]

| Q, M | P(A | Q, M) |
|------|-------------|
| f, f | 0.1 0.9     |
| f, t | 0.2 0.8     |
| t, f | 0.6 0.4     |
| t, t | 0.7 0.3     |
PRM: Aggregate Dependencies

Paper

- Quality
- Accepted

Review

- Mood
- Length

Review R1

- Mood

Review R2

- Mood

Review R3

- Mood
- Length

Paper P1

- Quality
- Accepted

© Getoor & Mihalkova 2010-2011
PRM: Aggregate Dependencies

<table>
<thead>
<tr>
<th>Q, M</th>
<th>( P(A \mid Q, M) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f, f )</td>
<td>0.1 0.9</td>
</tr>
<tr>
<td>( f, t )</td>
<td>0.2 0.8</td>
</tr>
<tr>
<td>( t, f )</td>
<td>0.6 0.4</td>
</tr>
<tr>
<td>( t, t )</td>
<td>0.7 0.3</td>
</tr>
</tbody>
</table>

- **Paper**
  - Quality
  - Accepted

- **Review**
  - Mood
  - Length

- **Paper P1**
  - Quality
  - Accepted

- **Review R1**
  - Mood

- **Review R2**
  - Mood

- **Review R3**
  - Mood

**sum, min, max, avg, mode, count**
PRM Semantics

PRM + relational skeleton $\sigma$ =

probability distribution over completions $l$:

$$P(l \mid \sigma, S, \Theta) = \prod_{x \in \sigma} \prod_{x.A} P(x.A \mid \text{parents}_{S,\sigma}(x.A))$$
Learning PRMs

- Parameter estimation
- Structure selection

© Getoor & Mihalkova 2010-2011
\[ \theta^* = \frac{N_{P,Q,R,M,P,A}}{N_{P,Q,R,M}} \]

where \( N_{P,Q,R,M,P,A} \) is the number of accepted, low quality papers whose reviewer was in a poor mood.
ML Parameter Estimation

\[ \theta^* = \frac{N_{P,\overline{Q},R,\overline{M},P,A}}{N_{P,\overline{Q},R,\overline{M}}} \]

Query for counts:

Count \[\pi\]  \[P.\text{Quality}\]  \[R.\text{Mood}\]  \[P.\text{Accepted}\]

© Getoor & Mihalkova 2010-2011
Structure Selection

Idea:
- define scoring function
- do local search over legal structures

Key Components:
- legal models
- scoring models
- searching model space
Structure Selection

Idea:
- define scoring function
- do local search over legal structures

Key Components:
- legal models
- scoring models
- searching model space
PRM defines a coherent probability model over a skeleton $\sigma$ if the dependencies between object attributes is acyclic.

How do we guarantee that a PRM is acyclic for every skeleton?
Attribute Stratification

Dependency graph acyclic $\Rightarrow$ acyclic for any $\sigma$

Algorithm more flexible; allows certain cycles along guaranteed acyclic relations
Structure Selection

Idea:
- define scoring function
- do local search over legal structures

Key Components:
- legal models
  - scoring models – same as BN
- searching model space
Structure Selection

Idea:
- define scoring function
- do local search over legal structures

Key Components:
- legal models
- scoring models
  » searching model space
Phase 0: consider only dependencies within a class

\[ \text{PotentialParents}(R.A) = \bigcup_{R.B \in \text{descriptive-attributes}(R)} R.B \]

© Getoor & Mihalkova 2010-2011
Phase 1: consider dependencies from “neighboring classes, via schema relations

\[ \text{Potential-Parents}(R.A) = \bigcup S.C \]

\[ S.C \in \text{descriptive-attributes}(R \rightarrow S) \]

© Getoor & Mihalkova 2010-2011
Phase 2: consider dependencies from “further” classes, via relation chains

$$\text{Potential-Parents}(R.A) = \bigcup T.D \quad T.D \in \text{descriptive-attributes}(R \triangleleft S \triangleleft T)$$
Reminder: PRM w/ AU Semantics

PRM + relational skeleton $\sigma = \prod_{x \in \sigma} \prod_{x.A} P(x.A \mid \text{parents}_{S,\sigma}(x.A))$

Objects Attributes

© Getoor & Mihalkova 2010-2011
Kinds of structural uncertainty

- How many objects does an object relate to?
  - how many Authors does Paper1 have?

- Which object is an object related to?
  - does Paper1 cite Paper2 or Paper3?

- Which class does an object belong to?
  - is Paper1 a JournalArticle or a ConferencePaper?

- Does an object actually exist?

- Are two objects identical?
Structural Uncertainty

- Motivation: PRM with AU only well-defined when the skeleton structure is known
- May be uncertain about relational structure itself
- Construct probabilistic models of relational structure that capture \textit{structural uncertainty}

- Mechanisms:
  - Reference uncertainty
  - Existence uncertainty
  - Number uncertainty
  - Type uncertainty
  - Identity uncertainty
Citation Relational Schema

Author
- Institution
- Research Area

Paper
- Topic
- Word1
- Word2
- ... WordN

Cites
- Citing Paper
- Cited Paper

Paper
- Topic
- Word1
- Word2
- ... WordN

© Getoor & Mihalkova 2010-2011
Attribute Uncertainty

Author

Institution

Research Area

Wrote

Paper

Topic

Word1

... WordN

P( Institution | Research Area)

P( Topic | Paper.Author.Research Area)

P( WordN | Topic)
Reference Uncertainty

Scientific Paper

Bibliography
1. -----
2. -----
3. -----?

Document Collection

© Getoor & Mihalkova 2010-2011
Dependency model for foreign keys

Naïve Approach: multinomial over primary key
  • noncompact
  • limits ability to generalize
Reference Uncertainty Example

- Paper P1 Topic Theory
- Paper P5 Topic AI
- Paper P3 Topic AI
- Paper P4 Topic Theory
- Paper P2 Topic Theory
- Paper P1 Topic Theory

Cites

Citing Cited

C1 C2

0.3 0.7

© Getoor & Mihalkova 2010-2011
Introduce Selector RV, whose domain is \{C1,C2\}
The distribution over Cited depends on all of the topics, and the selector
PRMs w/ RU Semantics

PRM RU + entity skeleton $\sigma$

$\Rightarrow$ probability distribution over full instantiations I
Existence Uncertainty

Document Collection

© Getoor & Mihalkova 2010-2011
Dependency model for existence of relationship
Exists Uncertainty Example

<table>
<thead>
<tr>
<th>Citer.Topic</th>
<th>Cited.Topic</th>
<th>False</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theory</td>
<td>Theory</td>
<td>0.995</td>
<td>0005</td>
</tr>
<tr>
<td>Theory</td>
<td>AI</td>
<td>0.999</td>
<td>0001</td>
</tr>
<tr>
<td>AI</td>
<td>Theory</td>
<td>0.997</td>
<td>0003</td>
</tr>
<tr>
<td>AI</td>
<td>AI</td>
<td>0.993</td>
<td>0008</td>
</tr>
</tbody>
</table>
Introduce Exists RVs

© Getoor & Mihalkova 2010-2011
Introduce Exists RVs
PRMs w/ EU Semantics

PRM-EU + object skeleton $\sigma$

$\Rightarrow$ probability distribution over full instantiations

© Getoor & Mihalkova 2010-2011
PRMs with classes

- Relations organized in a class hierarchy
  
  ![Class Hierarchy Diagram](Venue\(\backslash\)Journal\(\backslash\)Conference)

- Subclasses inherit their probability model from superclasses
- Instances are a special case of subclasses of size 1
- As you descend through the class hierarchy, you can have richer dependency models
  - e.g. cannot say `Accepted(P1) \(-\) Accepted(P2)` (cyclic)
  - but can say `Accepted(JournalP1) \(-\) Accepted(ConfP2)`
PRMs w/ Class Hierarchies

Allow us to:

- Refine a “heterogenous” class into more coherent subclasses
- Refine probabilistic model along class hierarchy
  - Can specialize/inherit CPDs
  - Construct new dependencies that were originally “acyclic”

*Provides bridge from class-based model to instance-based model*
Summary: PRMs

- Focus on objects and relationships
  - what types of objects are there, and how are they related to each other?
  - how does a property of an object depend on other properties (of the same or other objects)?

- Representation support
  - Attribute uncertainty
  - Structural uncertainty
  - Class Hierarchies

- Efficient Inference and Learning Algorithms
Road Map

Factor Graphs

Bayesian Nets

Markov Nets

Par-factor Graphs

Directed Models

BLPs [Kersting & De Raedt, ILP01]
PRMs [Koller & Pfeffer, AAAI98]
etc.

Undirected Models

RMNs [Taskar et al., UAI02]
MLNs [Richardson & Domingos, MLJ06]
PSL [Bröcheler et al., UAI10]
etc.

Hybrid Models

RDNs [Neville & Jensen, JMLR07]
Undirected Models

- Par-factor graphs that instantiate into undirected graphical models
- Will talk about
  - Markov logic networks
    - [Richardson & Domingos, MLJ06]
  - Probabilistic soft logic
    - [Bröcheler et al., UAI10]
Markov Logic Networks

- Par-factors are defined using first-order logic rules

\[
\text{hyperlink}(D_1, D_2) \rightarrow \text{category}(D_1, C) \land \text{category}(D_2, C)
\]

The predicates whose values are known during inference can be seen as constraining the cliques that are constructed over the unknown ones.

\[
A = \{\text{category}(D_1, C), \text{category}(D_2, C)\}
\]

© Getoor & Mihalkova 2010-2011
Markov Logic Networks

- Par-factors are defined using first-order logic statements

\[ \phi = \exp(w \cdot \text{TruthValue}(f(A))) \]
Instantiating the MLN

**Given rule:** \( \text{hyperlink}(D_1, D_2) \Rightarrow \text{category}(D_1, C) \land \text{category}(D_2, C) \)

**Given evidence:**

\[
\text{hyperlink}( \Box, \Box ) \quad \text{category}( \Box, \bullet ) \\
\text{hyperlink}( \Box, \Box ) \quad \text{category}( \Box, \leftarrow ) \\
\text{hyperlink}( \Box, \Box ) \\
\text{hyperlink}( \Box, \Box ) \\
\text{hyperlink}( \Box, \Box ) \\
\text{hyperlink}( \Box, \Box ) \\
\text{hyperlink}( \Box, \Box ) \\
\text{hyperlink}( \Box, \Box ) \\
\text{hyperlink}( \Box, \Box ) \\. \ldots \text{ etc.}
\]
Improving Efficiency

- Replacing known values into the instantiated rules

\[ \text{hyperlink}( \text{False}, \text{True} ) \rightarrow \text{category}( \text{True}, \text{True} ) \wedge \text{category}( \text{True}, \text{True} ) \]

Rule is trivially satisfied and can be safely ignored

... etc.
Improving Efficiency

- Removing rule instantiations that are trivially satisfied
Improving Efficiency

- Simplifying formulas
Improving Efficiency

- Now suppose we are interested in the truth value of just one document-category pair.
- What do we need to consider?

Irrelevant to our query!
Improving Efficiency

- Suppose in addition that we obtain new evidence:

  - This is a simple example of knowledge-based model construction [Wellman et al. KER92]
MLNs Joint Distribution

\[
X = \begin{cases} 
\text{Category}(D_1, C_1) \\
\text{Category}(D_1, C_2) \\
\vdots \\
\text{Category}(D_2, C_1) \\
\text{Category}(D_2, C_2) \\
\vdots \\
\text{Category}(D_{100}, C_1) \\
\vdots 
\end{cases}
\]

\[
P(X = x) = \frac{\exp \left( \sum_{f_i \in \mathcal{F}} w_i n_i(x) \right)}{Z}
\]

Possible world

Number of satisfied instantiations

For each formula in the MLN

© Getoor & Mihalkova 2010-2011
Lazy Inference

Insight: most relational domains are sparse
- i.e. a very small portion of all possible relations is actually realized

In lazy inference:
- Not all instantiations of the unknown predicates are formed up front
- For instantiations that are not formed, a default value is assumed
- Instantiations are formed when there is a chance that their truth value deviates from the default
Cutting Plane Inference

- Used to find most likely joint assignment
- Exploits a similar idea
  - Most rule instantiations will be satisfied due to the large number of default valued predicate instantiations
- Cutting plane inference
  - Convert the set of rule instantiations into an integer linear program
  - Only include instantiations that are not yet satisfied by the current assignment
  - Solve, and iterate
MLN Structure Learning

- **Goal:** Automatically induce an effective set of rules from data
- **Two flavors:**
  - **Top-down learning**
    - Start from a set of one-literal rules
    - In each iteration, attempt all possible ways of extending current rules, keeping the best k
    - Rules are evaluated according to a probabilistic score
  - **Bottom-up learning**
    - Use a data-driven procedure to focus search for candidate structures to more promising regions
MLN Structure Learning

Will discuss according to several dimensions:

- Transfer/Scratch
  - Transfer: Algorithm requires previously learned model
  - Scratch: Algorithm does not need transferred model

- Discriminative?
  - Algorithm requires knowledge about test attributes/relations

- Top-Down/Bottom-Up
## MLN Structure Learning

<table>
<thead>
<tr>
<th>Reference</th>
<th>Scratch/Transfer</th>
<th>Discriminative?</th>
<th>Top-Down/Bottom-UP</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Kok &amp; Domingos, ICML05]</td>
<td></td>
<td></td>
<td>Top-Down</td>
</tr>
<tr>
<td>[Mihalkova &amp; Mooney, ICML07]</td>
<td>Scratch</td>
<td></td>
<td>Bottom-Up</td>
</tr>
<tr>
<td>[Mihalkova &amp; Mooney, AAAI07]</td>
<td>Transfer</td>
<td></td>
<td>Bottom-Up</td>
</tr>
<tr>
<td>[Huynh &amp; Mooney, ICML08]</td>
<td>Scratch</td>
<td>Yes</td>
<td>Bottom-Up</td>
</tr>
<tr>
<td>[Biba et al., ILP08]</td>
<td></td>
<td>Yes</td>
<td>Top-Down</td>
</tr>
<tr>
<td>[Davis &amp; Domingos, ICML09]</td>
<td>Transfer</td>
<td></td>
<td>Top-Down</td>
</tr>
<tr>
<td>[Kok &amp; Domingos, ICML09]</td>
<td>Scratch</td>
<td></td>
<td>Bottom-Up</td>
</tr>
<tr>
<td>[Kok &amp; Domingos, ICML10]</td>
<td>Scratch</td>
<td></td>
<td>Bottom-Up</td>
</tr>
<tr>
<td>[Khorsavi et al., AAAI10]</td>
<td>Scratch</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Blank means that algorithm can be adapted either way

© Getoor & Mihalkova 2010-2011
Bottom-Up Idea Nuggets

- BUSL [Mihalkova & Mooney, ICML07]
  - Observation: An MLN defines a Markov net, so why not:
    - Start from learning a Markov network template
    - Constrain structure search to this template

- Algorithm of [Huynh & Mooney, ICML08]
  - Use a bottom-up ILP learner (ALEPH) to induce a set of clauses

- LHL [Kok & Domingos, ICML09]
  - Observation: Relational pathfinding is very effective in finding long-range clauses, but can blow up:
    - Cluster the entities in the domain
    - Perform relational pathfinding in the clustered graph

- LSM [Kok & Domingos, ICML10]
  - Want to learn even longer-range dependencies
    - Perform a random walk on the relational graph to find well-treaded patterns (structural motifs)
    - Constrain structure search to within motifs
Undirected Models

- Par-factor graphs that instantiate into undirected graphical models
- Will talk about
  - Markov logic networks
    - [Richardson & Domingos, MLJ06]
  - Probabilistic soft logic
    - [Bröcheler et al., UAI10]
Probabilistic Soft Logic

- Want to incorporate “soft” truth values
  - In the evidence
  - In the conclusions

- Want to reason about properties aggregated over sets

- Need to do all of this efficiently
  - Changes inference from solving an integer linear program to continuous optimization problem

© Getoor & Mihalkova 2010-2011
Probabilistic Soft Logic

- Par-factors are defined using rules consisting of logic-like & object-oriented syntax
- A PSL program instantiates to a continuous-valued Markov network

Following slides on PSL adapted from slides by Matthias Bröcheler
PSL Representation

- Can express general knowledge about relational domains

\[
\text{Category}(A, C) \iff \text{Category}(B, C) \land \text{Unknown}(A) \\
\land \text{link}(A, B) \land A \neq B
\]
Example: Voter Opinion Modeling

\[ \text{vote}(A,P) \land \text{friend}(B,A) \rightarrow \text{vote}(B,P) : 0.3 \]

\[ \text{vote}(A,P) \land \text{spouse}(B,A) \rightarrow \text{vote}(B,P) : 0.8 \]
PSL Representation

- Can plug in arbitrary similarity functions

\[ A \approx B \iff A\.name \approx B\.name \]

- Maximum flexibility for attribute similarity
- Customization to particular problem domains
PSL Representation

- Can incorporate relation-defined sets and similarity measures over sets

\[ A \approx B \iff \{A\text{.friends}\} \approx \{B\text{.friends}\} \]

- Users can define their own

- PSL comes with some predefined measures
Predefined Set Measures

- Average similarity between pairs of elements

\[ X \approx Y = \frac{\sum_{x \in X} \sum_{y \in Y} x \approx y}{|X||Y|} \]

- If for each \( x \), total sum of similarities to any \( y \) is at most 1:

\[ X \approx Y = \frac{2 \sum_{x \in X} \sum_{y \in Y} x \approx y}{|X| + |Y|} \]
Combining Soft Values in PSL

\[ H_1 \lor H_2 \lor \cdots \lor H_m \leftarrow B_1 \land B_2 \land \cdots \land B_n \]

- Soft values in a rule are combined using T-norms:
  - Lukasiewicz T-norm (can be customized)
    - \( \lor = \oplus(h_1, h_2) = \min(1, h_1 + h_2) \)
    - \( \land = \otimes(h_1, h_2) = \max(0, h_1 + h_2 - 1) \)
Distance to Satisfaction

\[ H_1 \lor H_2 \lor \cdots \lor H_m \iff B_1 \land B_2 \land \cdots \land B_n \]

- Establish satisfaction

\[ \oplus(H_1, \ldots, H_m) \geq \otimes(B_1, \ldots, B_n) \]

\[ R \approx T:? \iff A \approx B:0.7 \land D \approx E:0.8 \]
Distance to Satisfaction

\[ H_1 \lor H_2 \lor \cdots \lor H_m \iff B_1 \land B_2 \land \cdots \land B_n \]

- Establish satisfaction

\[ \oplus(H_1, \ldots, H_m) \geq \otimes(B_1, \ldots, B_n) \]

\[ R \approx T: \geq 0.5 \iff A \approx B: 0.7 \land D \approx E: 0.8 \]
Distance to Satisfaction

\[ H_1 \lor H_2 \lor \cdots \lor H_m \iff B_1 \land B_2 \land \cdots \land B_n \]

- Distance to satisfaction

\[
\max(\bigotimes(B_1, \ldots, B_n) - \bigoplus(H_1, \ldots, H_m), 0)
\]

\[
\begin{array}{c}
R \approx T: 0.7 \iff A \approx B: 0.7 \land D \approx E: 0.8 \\
R \approx T: 0.2 \iff A \approx B: 0.7 \land D \approx E: 0.8 \\
\end{array}
\]

© Getoor & Mihalkova 2010-2011
Rule Weights

\[ H_1 \lor H_2 \lor \cdots \lor H_m \iff B_1 \land B_2 \land \cdots \land B_n \]

- Weighted distance to satisfaction

\[ d(R, I) = w \times \max(\otimes(B_1, \ldots, B_n) - \oplus(H_1, \ldots, H_m), 0) \]
Rule Weights

\[ \begin{align*}
H_1 \lor H_2 \lor \cdots \lor H_m & \leftarrow B_1 \land B_2 \land \cdots \land B_n \\
\text{W}
\end{align*} \]

- Weighted distance to satisfaction

\[ d(R, I) = w \times \max \left( \otimes (B_1, \ldots, B_n) - \oplus (H_1, \ldots, H_m), 0 \right) \]

- Every rule instantiation \( R \) contributes a factor \( \phi_R = D(R, I) \) to the corresponding continuous-valued Markov network

© Getoor & Mihalkova 2010-2011
PSL Joint Distribution

\[ X = \left\{ \begin{array}{c}
\text{Category}(D1, C1) \\
\text{Category}(D1, C2) \\
\vdots \\
\text{Category}(D2, C1) \\
\text{Category}(D2, C2) \\
\vdots \\
\text{Category}(D100, C1) \\
\vdots
\end{array} \right\} \]

\[ f(x) = \frac{1}{Z} \exp[-\sum_{j=1}^{m} \phi_j(x)] \]

Normalizing constant

Weighted distance to satisfaction

Sum over all instantiations of all PSL rules

**© Getoor & Mihalkova 2010-2011**
Efficient Inference in PSL

- Attribute and set similarity functions computed as “black boxes”
- A PSL program instantiates to a continuous-valued Markov network
- Supports both
  - Finding most likely joint assignment
  - Computing marginals
Inference: Most Likely Assignment

- Inference is cast as a constrained continuous numerical optimization problem, solved in polynomial time
  - Convex optimization problem due to our choices of combination functions
  - $O(n^{3.5})$ inference
    - $n=$number of (active) instantiated rules
  - Rules are instantiated on an as-needed basis only when it is not maximally satisfied by current truth assignment
- This is in contrast to models where truth values are constrained to be either 0 or 1

© Getoor & Mihalkova 2010-2011
Inference: Computing Marginals

- Exact inference is \#P-hard

- Done using an MCMC sampling technique
  - Provable polynomial-time approximation guarantees
  - Uses a hit-and-run sampling scheme

- Can interpret as confidence
Learning in PSL

- Weight learning done in standard way
- Structure learning: currently in progress
Wikipedia Rules

\[ \text{hasCat}(A, C) \iff \text{hasCat}(B, C) \land A \neq B \land \text{unknown}(A) \land \text{similarText}(A, B) \]

\[ \text{hasCat}(A, C) \iff \text{hasCat}(B, C) \land \text{unknown}(A) \land \text{link}(A, B) \land A \neq B \]

\[ \text{hasCat}(D, C) \iff \text{talk}(D, A) \land \text{talk}(E, A) \land \text{hasCat}(E, C) \land \text{unknown}(D) \land A \neq B \]

© Getoor & Mihalkova 2010-2011
Wikipedia Results

![Graph showing F1 score for various attributes and document sizes.]

© Getoor & Mihalkova 2010-2011
Summary So Far

- Started with relational classifiers
  - Focus: relational feature construction
- Moved to collective classification models
  - Focus: propagating label assignments
- Considered advanced SRL languages
  - Focus: representing shared structure while allowing for principled learning and inference