Practical Statistical Relational AI

Pedro Domingos
Dept. of Computer Science & Eng.
University of Washington

With slight deletions and additions by Lily Mihalkova
Also using slides from…

Round and Efficient Inference with Probabilistic and Deterministic Dependencies

Hoifung Poon
University of Washington

(Joint work with Pedro Domingos)
A General Method for Reducing the Complexity of Relational Inference And its Application to MCMC

Hoifung Poon
University of Washington

(Joint work with Pedro Domingos and Marc Sumner)
The real world is complex and uncertain
Logic handles complexity
Probability handles uncertainty
Overview

Motivation

Foundational areas
- Probabilistic inference
- Statistical learning
- Logical inference
- Inductive logic programming

Putting the pieces together

Applications
Markov Networks

**Undirected** graphical models

\[
P(x) = \frac{1}{Z} \prod_c \Phi_c(x_c)
\]

\[
Z = \sum_x \prod_c \Phi_c(x_c)
\]

<table>
<thead>
<tr>
<th>Smoking</th>
<th>Cancer</th>
<th>Φ(S,C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>4.5</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>4.5</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>2.7</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>4.5</td>
</tr>
</tbody>
</table>
Markov Networks

Undirected graphical models

Smoking  Cancer  Asthma  Cough

Log-linear model:

\[ P(x) = \frac{1}{Z} \exp \left( \sum_i w_i f_i(x) \right) \]

\[ f_1(\text{Smoking, Cancer}) = \begin{cases} 
1 & \text{if } \neg \text{Smoking } \lor \text{Cancer} \\
0 & \text{otherwise}
\end{cases} \]

\[ w_1 = 1.5 \]
Famersley-Clifford Theorem

Distribution is strictly positive \( (P(x) > 0) \)

\textbf{and} Graph encodes conditional independences

\textbf{then} Distribution is product of potentials over cliques of graph

\textbf{verse} is also true.
Inference in Markov Networks

**Goal:** Compute marginals & conditionals of

\[ P(X) = \frac{1}{Z} \exp \left( \sum_i w_i f_i(X) \right) \]

\[ Z = \sum_x \exp \left( \sum_i w_i f_i(X) \right) \]

Exact inference is \#P-complete

Conditioning on Markov blanket is easy:

\[ (x \mid MB(x)) = \frac{\exp \left( \sum_i w_i f_i(x) \right)}{\exp \left( \sum_i w_i f_i(x = 0) \right) + \exp \left( \sum_i w_i f_i(x = 1) \right)} \]

Gibbs sampling exploits this
MCMC Gibb Sampling

state ← random truth assignment

for i ← 1 to num-samples do
    for each variable x
        sample x according to P(x|neighbors(x))
        state ← state with new value of x
    P(F) ← fraction of states in which F is true
Other Inference Methods

Many variations of MCMC
Belief propagation (sum-product)
Variational approximation
Exact methods
**Goal:** Find most likely state of world given evidence

$$\max_y P(y | x)$$

- **Query**
- **Evidence**
MAP Inference Algorithms

Iterated conditional modes
Simulated annealing
Graph cuts
Belief propagation (max-product)
Overview

Motivation

Foundational areas
- Probabilistic inference
- **Statistical learning**
- Logical inference
- Inductive logic programming

Putting the pieces together

Applications
Learning Markov Networks

Learning parameters (weights)
- Generatively
- Discriminatively

Learning structure (features)

In this tutorial: Assume complete data
(If not: EM versions of algorithms)
**Generative Weight Learning**

Maximize likelihood or posterior probability

Numerical optimization (gradient or 2nd order)

No local maxima

\[ \frac{\partial}{\partial w_i} \log P_w(x) = n_i(x) - E_w[n_i(x)] \]

- No. of times feature \( i \) is true in data
- Expected no. times feature \( i \) is true according to model

Requires inference at each step (slow!)
seudo-Likelihood

\[ PL(x) \equiv \prod_{i} P(x_i \mid \text{neighbors}(x_i)) \]

Likelihood of each variable given its neighbors in the data
Does not require inference at each step
Consistent estimator
Widely used in vision, spatial statistics, etc.
But PL parameters may not work well for long inference chains
discriminative Weight Learning

Maximize conditional likelihood of query \((y)\) given evidence \((x)\)

\[
\frac{\partial}{\partial w_i} \log P_w(y \mid x) = n_i(x, y) - E_w[n_i(x, y)]
\]

- No. of true groundings of clause \(i\) in data
- Expected no. true groundings according to model

Approximate expected counts by counts in MAP state of \(y\) given \(x\)
Structure Learning

Start with atomic features
Greedily conjoin features to improve score
Problem: Need to reestimate weights for each new candidate
Approximation: Keep weights of previous features constant
Overview

Motivation

Foundational areas
- Probabilistic inference
- Statistical learning
- **Logical inference**
- Inductive logic programming

Putting the pieces together

Applications
First-Order Logic

Constants, variables, functions, predicates
E.g.: Anna, x, MotherOf(x), Friends(x, y)

Literal: Predicate or its negation

Clause: Disjunction of literals

Grounding: Replace all variables by constants
E.g.: Friends (Anna, Bob)

World (model, interpretation):
Assignment of truth values to all ground predicates
Inference in First-Order Logic

Traditionally done by theorem proving (e.g.: Prolog)

Propositionalization followed by model checking turns out to be faster (often a lot)

**Propositionalization (or grounding):** Create all ground atoms and clauses

**Model checking:** Satisfiability testing

Two main approaches:
- **Backtracking** (e.g.: DPLL)
- **Stochastic local search** (e.g.: WalkSAT)
Input: Set of clauses
(Convert KB to conjunctive normal form (CNF))
Output: Truth assignment that satisfies all clauses, or failure

The paradigmatic NP-complete problem

Solution: Search

Key point:
Most SAT problems are actually easy

Hard region: Narrow range of #Clauses / #Variables
Stochastic Local Search

Uses complete assignments instead of partial
Start with random state
Flip variables in unsatisfied clauses
Hill-climbing: Minimize # unsatisfied clauses
Avoid local minima: Random flips
Multiple restarts
The WalkSAT Algorithm

\[\text{for } i \leftarrow 1 \text{ to } \text{max-tries} \text{ do} \]
\[\text{solution} = \text{random truth assignment} \]
\[\text{for } j \leftarrow 1 \text{ to } \text{max-flips} \text{ do} \]
\[\text{if all clauses satisfied then} \]
\[\text{return solution} \]
\[c \leftarrow \text{random unsatisfied clause} \]
\[\text{with probability } p \]
\[\text{flip a random variable in } c \]
\[\text{else} \]
\[\text{flip variable in } c \text{ that maximizes number of satisfied clauses} \]
\[\text{return failure} \]
Overview

Motivation

Foundational areas
- Probabilistic inference
- Statistical learning
- Logical inference
- Inductive logic programming

Putting the pieces together

Applications
**Rule Induction**

**Given:** Set of positive and negative examples of some concept

- **Example:** \((x_1, x_2, \ldots, x_n, y)\)
- **y:** concept (Boolean)
- \(x_1, x_2, \ldots, x_n:** attributes (assume Boolean)

**Goal:** Induce a set of rules that cover all positive examples and no negative ones

- **Rule:** \(x_a \land x_b \land \ldots \Rightarrow y\) (\(x_a:** Literal, i.e., \(x_i\) or its negation)
- Same as **Horn clause:** \(Body \Rightarrow Head\)
- Rule \(r\) covers example \(x\) iff \(x\) satisfies body of \(r\)

**Eval**\((r)\): Accuracy, info. gain, coverage, support, etc.
Learning a Single Rule

\[
\begin{align*}
\text{head} & \leftarrow y \\
\text{body} & \leftarrow \emptyset \\
\text{repeat} & \\
& \text{for each literal } x \\
& \quad r_x \leftarrow r \text{ with } x \text{ added to } \text{body} \\
& \quad \text{Eval}(r_x) \\
& \quad \text{body} \leftarrow \text{body} \cup \text{best } x \\
\text{until} & \text{ no } x \text{ improves } \text{Eval}(r) \\
\text{return } & r
\end{align*}
\]
learning a Set of Rules

\[
\begin{align*}
R & \leftarrow \emptyset \\
S & \leftarrow \text{examples} \\
\text{repeat} & \\
& \quad \text{learn a single rule } r \\
& \quad R \leftarrow R \cup \{ r \} \\
& \quad S \leftarrow S - \text{positive examples covered by } r \\
\text{until } & S \text{ contains no positive examples} \\
\text{return } & R
\end{align*}
\]
First-Order Rule Induction

\( y \) and \( x_i \) are now predicates with arguments
E.g.: \( y \) is Ancestor\((x,y)\), \( x_i \) is Parent\((x,y)\)

Literals to add are predicates or their negations
Literal to add must include at least one variable already appearing in rule
Adding a literal changes \# groundings of rule
E.g.: Ancestor\((x,z) \land Parent(z,y) \Rightarrow Ancestor(x,y)\)

Eval\((r)\) must take this into account
E.g.: Multiply by \# positive groundings of rule
still covered after adding literal
Overview

Motivation

Foundational areas
- Probabilistic inference
- Statistical learning
- Logical inference
- Inductive logic programming

Putting the pieces together

Applications
Iethora of Approaches

Knowledge-based model construction [Wellman et al., 1992]
Stochastic logic programs [Muggleton, 1996]
Probabilistic relational models [Friedman et al., 1999]
Relational Markov networks [Taskar et al., 2002]
Bayesian logic [Milch et al., 2005]
Markov logic [Richardson & Domingos, 2006]
And many others!
Key Dimensions

Logical language
First-order logic, Horn clauses, frame systems

Probabilistic language
Bayes nets, Markov nets, PCFGs

Type of learning
- Generative / Discriminative
- Structure / Parameters
- Knowledge-rich / Knowledge-poor

Type of inference
- MAP / Marginal
- Full grounding / Partial grounding / Lifted
Markov Logic

Logical language: First-order logic
Probabilistic language: Markov networks
- Syntax: First-order formulas with weights
- Semantics: Templates for Markov net features

Learning:
- Parameters: Generative or discriminative
- Structure: Variety of techniques

Inference:
- MAP: Weighted satisfiability, Cutting plane inference
- Marginal: MCMC with moves proposed by SAT solver, (lifted) Belief propagation
- Partial grounding + Lazy inference
Markov Logic: Intuition

A logical KB is a set of **hard constraints** on the set of possible worlds.

Let’s make them **soft constraints**: When a world violates a formula, it becomes less probable, not impossible.

Give each formula a **weight**:

(Higher weight ⇒ Stronger constraint)

\[ \text{probability}(\text{world}) \propto \exp\left( \sum \text{weights of formulas it satisfies} \right) \]
A Markov Logic Network (MLN) is a set of pairs $(F, w)$ where

- $F$ is a formula in first-order logic
- $w$ is a real number

Together with a set of constants, it defines a Markov network with

- One node for each grounding of each predicate in the MLN
- One feature for each grounding of each formula $F$ in the MLN, with the corresponding weight $w$
Example: Friends & Smokers

Smoking causes cancer.
Friends have similar smoking habits.
Example: Friends & Smokers

\[ \forall x \ Smokes(x) \Rightarrow Cancer(x) \]
\[ \forall x, y \ Friends(x, y) \Rightarrow (Smokes(x) \iff Smokes(y)) \]
Example: Friends & Smokers

<table>
<thead>
<tr>
<th></th>
<th>( \forall x \text{ Smokes}(x) \Rightarrow \text{Cancer}(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \forall x, y \text{ Friends}(x, y) \Rightarrow (\text{Smokes}(x) \Leftrightarrow \text{Smokes}(y)) )</td>
</tr>
</tbody>
</table>
Example: Friends & Smokers

1.5 $\forall x \text{Smokes}(x) \Rightarrow \text{Cancer}(x)$

1.1 $\forall x, y \text{Friends}(x, y) \Rightarrow (\text{Smokes}(x) \Leftrightarrow \text{Smokes}(y))$

Two constants: Anna (A) and Bob (B)
Example: Friends & Smokers

1.5 \( \forall x \text{ Smokes}(x) \Rightarrow \text{Cancer}(x) \)

1.1 \( \forall x, y \text{ Friends}(x, y) \Rightarrow (\text{Smokes}(x) \Leftrightarrow \text{Smokes}(y)) \)

Two constants: Anna (A) and Bob (B)
Example: Friends & Smokers

Two constants: Anna (A) and Bob (B)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5</td>
<td>$\forall x \ Smokes(x) \Rightarrow Cancer(x)$</td>
</tr>
<tr>
<td>1.1</td>
<td>$\forall x, y \ Friends(x, y) \Rightarrow (Smokes(x) \Leftrightarrow Smokes(y))$</td>
</tr>
</tbody>
</table>
Example: Friends & Smokers

1.5 \( \forall x \text{ Smokes}(x) \Rightarrow \text{Cancer}(x) \)

1.1 \( \forall x, y \text{ Friends}(x, y) \Rightarrow (\text{Smokes}(x) \Leftrightarrow \text{Smokes}(y)) \)

Two constants: **Anna** (A) and **Bob** (B)
Example: Friends & Smokers

1. \[ \forall x \text{ Smokes}(x) \Rightarrow \text{Cancer}(x) \]
2. \[ \forall x, y \text{ Friends}(x, y) \Rightarrow (\text{Smokes}(x) \Leftrightarrow \text{Smokes}(y)) \]

Two constants: Anna (A) and Bob (B)
Markov Logic Networks

MLN is template for ground Markov nets

Probability of a world $x$:

$$P(x) = \frac{1}{Z} \exp \left( \sum_{i} w_i n_i(x) \right)$$

- Weight of formula $i$
- No. of true groundings of formula $i$ in $x$

**Typed** variables and constants greatly reduce size of ground Markov net

Functions, existential quantifiers, etc.

Infinite and continuous domains
relation to First-Order Logic

Infinite weights $\Rightarrow$ First-order logic
Satisfiable KB, positive weights $\Rightarrow$
Satisfying assignments = Modes of distribution
Markov logic allows contradictions between formulas
Problem: Find most likely state of world given evidence

\[
\arg \max_y P(y \mid x)
\]
**MAP/MPE Inference**

**Problem:** Find most likely state of world given evidence

\[
\arg \max_y \frac{1}{Z_x} \exp \left( \sum_i w_i n_i (x, y) \right)
\]
Problem: Find most likely state of world given evidence

$$\text{arg max } \sum_{y} w_i n_i(x, y)$$
**MAP/MPE Inference**

**Problem:** Find most likely state of world given evidence

\[
\arg \max_y \sum_i w_i n_i(x, y)
\]

This is just the weighted MaxSAT problem

Use weighted SAT solver

(e.g., MaxWalkSAT [Kautz et al., 1997])

Potentially faster than logical inference (!)
The WalkSAT Algorithm

for $i \leftarrow 1$ to $\text{max-tries}$ do
  solution = random truth assignment
  for $j \leftarrow 1$ to $\text{max-flips}$ do
    if all clauses satisfied then
      return solution
    c $\leftarrow$ random unsatisfied clause
    with probability $p$
      flip a random variable in $c$
    else
      flip variable in $c$ that maximizes number of satisfied clauses
  return failure
The MaxWalkSAT Algorithm

for $i \leftarrow 1$ to $\text{max-tries}$ do
  $\text{solution} = \text{random truth assignment}$
  for $j \leftarrow 1$ to $\text{max-flips}$ do
    if $\sum \text{weights(sat. clauses)} > \text{threshold}$ then
      return $\text{solution}$
    $c \leftarrow \text{random unsatisfied clause}$
    with probability $p$
      flip a random variable in $c$
    else
      flip variable in $c$ that maximizes
      $\sum \text{weights(sat. clauses)}$
  return failure, best $\text{solution}$ found
but ... It Is Not Scalable

1000 researchers
Coauthor(x,y): 1 million ground atoms
Coauthor(x,y) ∧ Coauthor(y,z) ⇒ Coauthor(x,z):
1 billion ground clauses
Exponential in arity
parsity to the Rescue

000 researchers
\( \text{oauthor}(x,y) \): 1 million ground atoms

but … most atoms are false
\( \text{oauthor}(x,y) \land \text{Coauthor}(y,z) \Rightarrow \text{Coauthor}(x,z) \):

billion ground clauses

lost trivially satisfied if most atoms are false

so need to explicitly compute most of them
General Method for Lazy Inference

If most variables assume the default value, …

Wasteful to instantiate all variables / functions

Main idea:

- Allocate memory for a small subset of “active” variables / functions
- Activate more if necessary as inference proceeds

Reduce memory / time
Active Variables and Functions

Coa(A,A) ∧ Coa(A,A) ⇒ Coa(C,A)

Coa(A,B) ∧ Coa(B,C) ⇒ Coa(A,C)

Coa(C,A) ∧ Coa(A,B) ⇒ Coa(C,B)

Coa(C,B) ∧ Coa(B,A) ⇒ Coa(C,A)

Coa(C,B) ∧ Coa(B,B) ⇒ Coa(C,B)
Active Variables and Functions

Coa(C,A) \land Coa(A,A) \Rightarrow Coa(C,A)

Coa(A,B) \land Coa(B,C) \Rightarrow Coa(A,C)

Coa(C,A) \land Coa(A,B) \Rightarrow Coa(C,B)

Coa(C,B) \land Coa(B,A) \Rightarrow Coa(C,A)

Coa(C,B) \land Coa(B,B) \Rightarrow Coa(C,B)
Variables and Functions

\begin{align*}
\text{Coa}(C,A) \land \text{Coa}(A,A) & \Rightarrow \text{Coa}(C,A) \\
\text{Coa}(A,B) \land \text{Coa}(B,C) & \Rightarrow \text{Coa}(A,C) \\
\text{Coa}(C,A) \land \text{Coa}(A,B) & \Rightarrow \text{Coa}(C,B) \\
\text{Coa}(C,B) \land \text{Coa}(B,A) & \Rightarrow \text{Coa}(C,A) \\
\text{Coa}(C,B) \land \text{Coa}(B,B) & \Rightarrow \text{Coa}(C,B)
\end{align*}
Computing Probabilities

\[ P(\text{Formula}|\text{MLN},C) = ? \]

MCMC: Sample worlds, check formula holds

\[ P(\text{Formula}_1|\text{Formula}_2,\text{MLN},C) = ? \]

If \text{Formula}_2 = \text{Conjunction of ground atoms}

- First construct min subset of network necessary to answer query
- Then apply MCMC (or other)
Problem:
Deterministic dependencies break MCMC
Near-deterministic ones make it very slow

Solution:
Combine MCMC and WalkSAT
→ MC-SAT algorithm  [Poon & Domingos, 2006]
Auxiliary-Variable Methods

Main ideas:
- Use auxiliary variables to capture dependencies
- Turn difficult sampling into uniform sampling

Given distribution $P(x)$

$$f(x,u) = \begin{cases} 1, & \text{if } 0 \leq u \leq P(x) \\ 0, & \text{otherwise} \end{cases} \quad \Rightarrow \quad \int f(x,u) \, du = P(x)$$

Sample from $f(x,u)$, then discard $u$
Slice Sampling [Damien et al. 1999]
slice Sampling

Identifying the slice may be difficult

\[ P(x) = \frac{1}{Z} \prod_i \Phi_i(x) \]

Introduce an auxiliary variable \( u_i \) for each \( \Phi_i \)

\[ f(x, u_1, \ldots, u_n) = \begin{cases} 
1 & \text{if } 0 \leq u_i \leq \Phi_i(x) \\
0 & \text{otherwise}
\end{cases} \]
The MC-SAT Algorithm

Approximate inference for Markov logic

Use slice sampling in MCMC

- Auxiliary var. $u_i$ for each clause $C_i$: $0 \leq u_i \leq \exp(w_i f_i(x))$
- $C_i$ unsatisfied: $0 \leq u_i \leq 1$
  $$\Rightarrow \exp(w_i f_i(x)) \geq u_i$$ for any next state $x$
- $C_i$ satisfied: $0 \leq u_i \leq \exp(w_i)$
  $$\Rightarrow \text{With prob. } 1 - \exp(-w_i), \text{ next state } x \text{ must satisfy } C_i$$
  to ensure that $\exp(w_i f_i(x)) \geq u_i$
The MC-SAT Algorithm

Select random subset $M$ of satisfied clauses.

Larger $w_i \Rightarrow C_i$ more likely to be selected.

Hard clause ($w_i \rightarrow \infty$): Always selected.

Slice = States that satisfy clauses in $M$.

Sample uniformly from these.
The MC-SAT Algorithm

0) \( \leftarrow \) A random solution satisfying all hard clauses

\( r \) \( k \leftarrow 1 \) to num_samples

\( M \leftarrow \emptyset \)

forall \( C_i \) satisfied by \( X(k-1) \)

With prob. \( 1 - \exp(-w_i) \) add \( C_i \) to \( M \)

endfor

\( X(k) \leftarrow \) A uniformly random solution satisfying \( M \)

dfor
Alchemy

Open-source software including:
Full first-order logic syntax
Generative & discriminative weight learning
Structure learning
Weighted satisfiability and MCMC
Programming language features

alchemy.cs.washington.edu
Some Examples

Thanks to Galileo, Alchemy is installed on fireball:

- /fs/junkfood/namatag/shared/bin/infer
- /fs/junkfood/namatag/shared/bin/learnwts
- /fs/junkfood/namatag/shared/bin/learnstruct
- Running them with no parameters gives you options

Can get examples at

- /fs/junkfood/lily/ClassExamples
Sample: Unbiased coin flips

Type: \( \text{flip} = \{ 1, \ldots, 20 \} \)

Predicate: \( \text{Heads}(\text{flip}) \)

\[
P(\text{Heads}(f)) = \frac{\frac{1}{Z}e^0}{\frac{1}{Z}e^0 + \frac{1}{Z}e^0} = \frac{1}{2}
\]
Example: Biased coin flips

default, MLN includes unit clauses for all predicates
captures marginal distributions, etc.)
What happens if you put a negative weight?
How does varying the number of sampling steps affect the quality of the probabilities?
Data is a relational database
Closed world assumption (if not: EM)
Learning parameters (weights)
Learning structure (formulas)
Weight Learning

Parameter tying: Groundings of same clause

$$\frac{\partial}{\partial w_i} \log P_w(x) = n_i(x) - E_w[n_i(x)]$$

- No. of times clause $i$ is true in data
- Expected no. times clause $i$ is true according to MLN

Generative learning: Pseudo-likelihood

Discriminative learning: Cond. likelihood, use MC-SAT or MaxWalkSAT for inference
Logistic Regression

Logistic regression:

$$\log \left( \frac{P(C=1 | F=f)}{P(C=0 | F=f)} \right) = a + \sum b_i f_i$$

Type:

$$\text{obj} = \{ 1, \ldots, n \}$$

Query predicate:

$$C(\text{obj})$$

Evidence predicates:

$$F_i(\text{obj})$$

Formulas:

$$a \ C(x)$$

$$b_i \ F_i(x) \land C(x)$$

Resulting distribution:

$$P(C=c, F=f) = \frac{1}{Z} \exp \left( ac + \sum b_i f_i c \right)$$

Therefore:

$$\log \left( \frac{P(C=1 | F=f)}{P(C=0 | F=f)} \right) = \log \left( \frac{\exp(a + \sum b_i f_i)}{\exp(0)} \right) = a + \sum b_i f_i$$

Alternative form:

$$F_i(x) \Rightarrow C(x)$$
Learn weights for the formulas in voting-implication.mln. Compare the weights learned with implication vs conjunction formulas.

Experiment with
- a) writing your own more complex clauses
- b) using structure learning to learn more complex clauses (learnstruct executable)
**Hidden Markov Models**

\[
\text{obs} = \{ \text{Obs1, ... , ObsN} \} \\
\text{state} = \{ \text{St1, ... , StM} \} \\
\text{time} = \{ 0, \ldots , T \} \\
\]

\[
\text{state}(\text{state}!, \text{time}) \\
\text{obs}(\text{obs}!, \text{time}) \\
\text{state}(+s, 0) \\
\text{state}(+s, t) \Rightarrow \text{State}(+s', t+1) \\
\text{obs}(+o, t) \Rightarrow \text{State}(+s, t) \\
\]
Make a rule that relates next state to previous two states. Observe how inference time changes.
Does it make a difference if you run lazy inference (with –lazy option)?
Text Classification

page = \{ 1, \ldots, n \}
ord = \{ \ldots \}
opic = \{ \ldots \}

\text{topic}(\text{page}, \text{topic}!)
\text{asWord}(\text{page}, \text{word})

\text{Topic}(p, t)
\text{asWord}(p, +w) \Rightarrow \text{Topic}(p, +t)
Hypertext Classification

\[
\text{topic}(\text{page}, \text{topic}) \\
\text{asWord}(\text{page}, \text{word}) \\
\text{links}(\text{page}, \text{page}) \\
\text{asWord}(p, +w) \Rightarrow \text{Topic}(p, +t) \\
\text{topic}(p, t) \land \text{links}(p, p') \Rightarrow \text{Topic}(p', t)
\]

Information Retrieval

\[\text{ifQuery}(\text{word}) \land \text{HasWord}(\text{page}, \text{word}) \land \text{relevant}(\text{page})\]

\[\text{ifQuery}(w+) \land \text{HasWord}(p, +w) \Rightarrow \text{relevant}(p)\]

\[\text{relevant}(p) \land \text{Links}(p, p') \Rightarrow \text{relevant}(p')\]

Entity Resolution

Problem: Given database, find duplicate records

dToken(token, field, record)
meField(field, record, record)
meRecord(record, record)

dToken(+t, +f, r) ^ HasToken(+t, +f, r')
=> SameField(f, r, r')
meField(f, r, r') => SameRecord(r, r')
meRecord(r, r') ^ SameRecord(r', r'')
=> SameRecord(r, r'')
Entity Resolution

...can also resolve fields:

- `^EntityResolution\(\)\`
  - `\)\`
  - `\)\`
  - `\)\`

- `\)\`
  - `\)\`
  - `\)\`

- `\)\`
  - `\)\`
  - `\)\`

- `\)\`
  - `\)\`
  - `\)\`

Practical Tips

Add all unit clauses (the default)

Implications vs. conjunctions

Open/closed world assumptions

How to handle uncertain data:
\[ R(x, y) \Rightarrow R'(x, y) \]  (the “HMM trick”)

Controlling complexity

- Low clause arities
- Low numbers of constants
- Short inference chains

Use the simplest MLN that works

Cycle: Add/delete formulas, learn and test
Structure Learning

Generalizes feature induction in Markov nets

Any inductive logic programming approach can be used, but . . .

Goal is to induce any clauses, not just Horn

Evaluation function should be likelihood

Requires learning weights for each candidate

Turns out not to be bottleneck

Bottleneck is counting clause groundings

Solution: Subsampling
Structure Learning

**Initial state:** Unit clauses or hand-coded KB

**Operators:** Add/remove literal, flip sign

**Evaluation function:** Pseudo-likelihood + Structure prior

**Search:** Beam, shortest-first

[Kok & Domingos, 2005]
Learning Markov Logic Network Structure Via Hypergraph Lifting

Stanley Kok
Dept. of Computer Science and Eng.
University of Washington
Seattle, USA

Joint work with Pedro Domingos
Output: Relational DB

Probabilistic KB
Relational Pathfinding
Richards & Mooney, AAAI’92]

Find paths of linked ground atoms → formulas
Path ∧ conjunction that is true at least once
Exponential search space of paths
Restricted to short paths