

# Statistical Modeling Of Relational Data

**Pedro Domingos**

Dept. of Computer Science & Eng.  
University of Washington



## Overview

- Motivation
- Foundational areas
  - Probabilistic inference
  - Statistical learning
  - Logical inference
  - Inductive logic programming
- Putting the pieces together
- Applications



## Motivation



Traditional KDD	Real World
Single relation	Multiple relations
Independent objects (i.i.d. data)	Interdependent objects (non-i.i.d. data)
One type of data	Multiple types of data
Pre-processing already done	Pre-processing is key problem
Knowledge-poor	Knowledge-rich

## Examples



- Web search
- Information extraction
- Natural language processing
- Perception
- Medical diagnosis
- Computational biology
- Social networks
- Ubiquitous computing
- Etc.

## Costs and Benefits of Multi-Relational Data Mining



- **Benefits**
  - Better predictive accuracy
  - Better understanding of domains
  - Growth path for KDD
- **Costs**
  - Learning is much harder
  - Inference becomes a crucial issue
  - Greater complexity for user

## Goal and Progress



- **Goal:**  
**Learn from multiple relations as easily as from a single one**
- Progress to date
  - Burgeoning research area
  - We're "close enough" to goal
  - Easy-to-use open-source software available
- Lots of research questions (old and new)



## Plan

- We have the elements:
  - **Probability** for handling uncertainty
  - **Logic** for representing types, relations, and complex dependencies between them
  - **Learning** and **inference** algorithms for each
- Figure out how to put them together
- Tremendous leverage on a wide range of applications



## Disclaimers

- Not a complete survey of multi-relational data mining
- Or of foundational areas
- Focus is practical, not theoretical
- Assumes basic background in logic, probability and statistics, etc.
- Please ask questions
- Tutorial and examples available at **[alchemy.cs.washington.edu](http://alchemy.cs.washington.edu)**



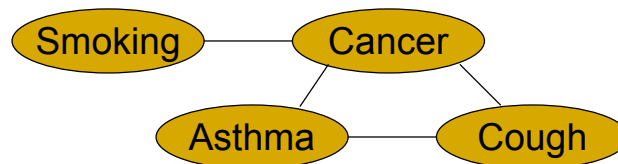
## Overview

- Motivation
- **Foundational areas**
  - **Probabilistic inference**
  - Statistical learning
  - Logical inference
  - Inductive logic programming
- Putting the pieces together
- Applications

## Markov Networks



- **Undirected** graphical models



- Potential functions defined over cliques

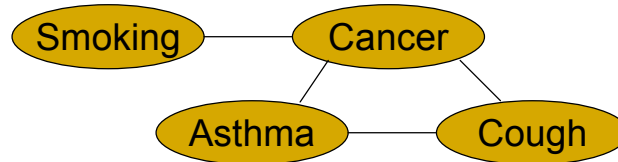
$$P(x) = \frac{1}{Z} \prod_c \Phi_c(x_c)$$

$$Z = \sum_x \prod_c \Phi_c(x_c)$$

Smoking	Cancer	$\Phi(S,C)$
False	False	4.5
False	True	4.5
True	False	2.7
True	True	4.5

## Markov Networks

- Undirected graphical models



- Log-linear model:

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

Weight of Feature  $i$       Feature  $i$

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

$$w_1 = 1.5$$

## Markov Nets vs. Bayes Nets

Property	Markov Nets	Bayes Nets
Form	Prod. potentials	Prod. potentials
Potentials	Arbitrary	Cond. probabilities
Cycles	Allowed	Forbidden
Partition func.	$Z = ?$	$Z = 1$
Indep. check	Graph separation	D-separation
Indep. props.	Some	Some
Inference	MCMC, BP, etc.	Convert to Markov

## Inference in Markov Networks



- **Goal:** Compute marginals & conditionals of

$$P(X) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(X)\right) \quad Z = \sum_X \exp\left(\sum_i w_i f_i(X)\right)$$

- Exact inference is #P-complete
- Conditioning on Markov blanket is easy:

$$P(x | MB(x)) = \frac{\exp\left(\sum_i w_i f_i(x)\right)}{\exp\left(\sum_i w_i f_i(x=0)\right) + \exp\left(\sum_i w_i f_i(x=1)\right)}$$

- Gibbs sampling exploits this

## MCMC: Gibbs 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



## MAP/MPE Inference

- **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 2<sup>nd</sup> 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!)



## Pseudo-Likelihood

$$PL(x) \equiv \prod_i P(x_i | \text{neighbors}(x_i))$$

- 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 ( $x$ )

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

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  $x$

## Other Weight Learning Approaches



- **Generative:** Iterative scaling
- **Discriminative:** Max margin

## 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:**  
Create all ground atoms and clauses
- **Model checking:** Satisfiability testing
- Two main approaches:
  - **Backtracking** (e.g.: DPLL; not covered here)
  - **Stochastic local search** (e.g.: WalkSAT)

## Satisfiability



- **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



```
for  $i \leftarrow 1$  to max-tries do  
  solution = random truth assignment  
  for  $j \leftarrow 1$  to 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
```



## 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:**  $(x_1, x_2, \dots, x_n, y)$
  - $y$ : **concept** (Boolean)
  - $x_1, x_2, \dots, x_n$ : **attributes** (assume Boolean)
- **Goal:** Induce a set of rules that cover all positive examples and no negative ones
  - **Rule:**  $x_a \wedge x_b \wedge \dots \Rightarrow y$  ( $x_a$ : 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



```
head ← y
body ← ∅
repeat
  for each literal x
    rx ← r with x added to body
    Eval(rx)
    body ← body ^ best x
until no x improves Eval(r)
return r
```

## Learning a Set of Rules



```
R ← ∅
S ← examples
repeat
  learn a single rule r
  R ← R ∪ {r}
  S ← S - positive examples covered by r
until S = ∅
return R
```

## 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**( $x,z$ )  $\wedge$  **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

## Plethora 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

## Knowledge-Based Model Construction



- **Logical language:** Horn clauses
- **Probabilistic language:** Bayes nets
  - Ground atom → Node
  - Head of clause → Child node
  - Body of clause → Parent nodes
  - >1 clause w/ same head → Combining function
- **Learning:** ILP + EM
- **Inference:** Partial grounding + Belief prop.

## Stochastic Logic Programs



- **Logical language:** Horn clauses
- **Probabilistic language:** Probabilistic context-free grammars
  - Attach probabilities to clauses
  - $\sum$  Probs. of clauses w/ same head = 1
- **Learning:** ILP + “Failure-adjusted” EM
- **Inference:** Do all proofs, add probs.

## Probabilistic Relational Models



- **Logical language:** Frame systems
- **Probabilistic language:** Bayes nets
  - Bayes net template for each class of objects
  - Object's attrs. can depend on attrs. of related objs.
  - Only binary relations
  - No dependencies of relations on relations
- **Learning:**
  - Parameters: Closed form (EM if missing data)
  - Structure: "Tiered" Bayes net structure search
- **Inference:** Full grounding + Belief propagation

## Relational Markov Networks



- **Logical language:** SQL queries
- **Probabilistic language:** Markov nets
  - SQL queries define cliques
  - Potential function for each query
  - No uncertainty over relations
- **Learning:**
  - Discriminative weight learning
  - No structure learning
- **Inference:** Full grounding + Belief prop.

## Bayesian Logic



- **Logical language:** First-order semantics
- **Probabilistic language:** Bayes nets
  - BLOG program specifies how to generate relational world
  - Parameters defined separately in Java functions
  - Allows unknown objects
  - May create Bayes nets with directed cycles
- **Learning:** None to date
- **Inference:**
  - MCMC with user-supplied proposal distribution
  - Partial grounding

## 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:** ILP with arbitrary clauses and MAP score
- **Inference:**
  - **MAP:** Weighted satisfiability
  - **Marginal:** MCMC with moves proposed by SAT solver
  - Partial grounding + Lazy inference

## Markov Logic



- Most developed approach to date
- Many other approaches can be viewed as special cases
- Main focus of rest of this tutorial

## 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  $\Rightarrow$  Stronger constraint)

$$P(\text{world}) \propto \exp\left(\sum \text{weights of formulas it satisfies}\right)$$

## Markov Logic: Definition



- 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



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

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

## Example: Friends & Smokers



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

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

## 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)

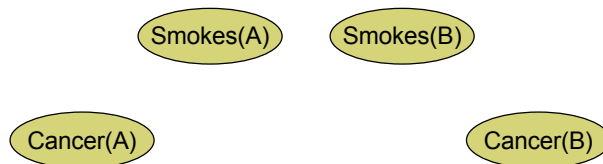
## Example: Friends & Smokers



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)



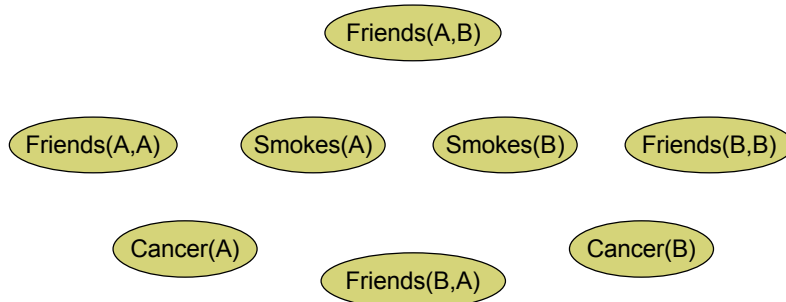
## Example: Friends & Smokers



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)



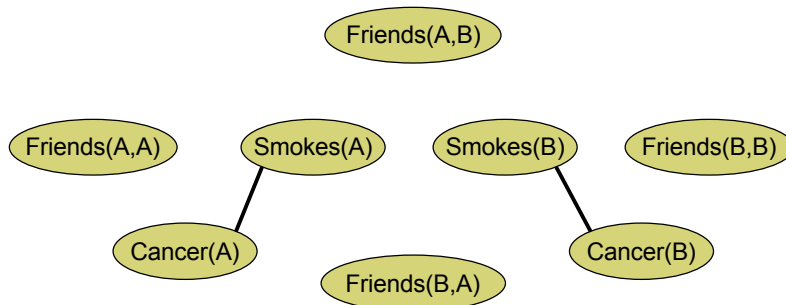
## Example: Friends & Smokers



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)



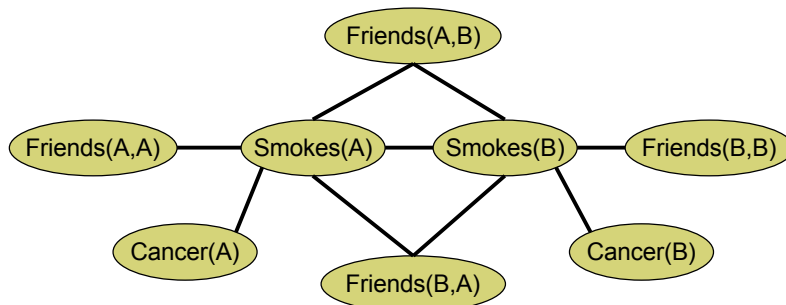
## Example: Friends & Smokers



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)



## 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 Statistical Models



- Special cases:
  - Markov networks
  - Markov random fields
  - Bayesian networks
  - Log-linear models
  - Exponential models
  - Max. entropy models
  - Gibbs distributions
  - Boltzmann machines
  - Logistic regression
  - Hidden Markov models
  - Conditional random fields
- Obtained by making all predicates zero-arity
- Markov logic allows objects to be interdependent (non-i.i.d.)

## Relation to First-Order Logic

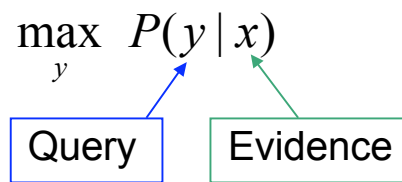


- Infinite weights  $\Rightarrow$  First-order logic
- Satisfiable KB, positive weights  $\Rightarrow$  Satisfying assignments = Modes of distribution
- Markov logic allows contradictions between formulas



## MAP/MPE Inference

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



## MAP/MPE Inference

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

$$\max_y \frac{1}{Z_x} \exp\left(\sum_i w_i n_i(x, y)\right)$$

## MAP/MPE Inference



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

$$\max_y \sum_i w_i n_i(x, y)$$

## MAP/MPE Inference



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

$$\max_y \sum_i w_i n_i(x, y)$$

- This is just the weighted MaxSAT problem
- Use weighted SAT solver (e.g., MaxWalkSAT [Kautz et al., 1997] )
- Potentially faster than logical inference (!)

## The MaxWalkSAT Algorithm



```
for  $i \leftarrow 1$  to  $max\text{-tries}$  do
   $solution$  = random truth assignment
  for  $j \leftarrow 1$  to  $max\text{-flips}$  do
    if  $\sum weights(\text{sat. clauses}) > \text{threshold}$  then
      return  $solution$ 
     $c \leftarrow$  random unsatisfied clause
    with probability  $p$ 
      flip a random variable in  $c$ 
    else
      flip variable in  $c$  that maximizes
         $\sum weights(\text{sat. clauses})$ 
  return failure, best  $solution$  found
```

## But ... Memory Explosion



- **Problem:**  
If there are  $n$  constants  
and the highest clause arity is  $c$ ,  
the ground network requires  $O(n^c)$  memory
- **Solution:**  
Exploit sparseness; ground clauses lazily  
→ LazySAT algorithm [Singla & Domingos, 2006]

## Computing Probabilities



- $P(\text{Formula}|\text{MLN},C) = ?$
- MCMC: Sample worlds, check formula holds
- $P(\text{Formula1}|\text{Formula2},\text{MLN},C) = ?$
- If **Formula2** = Conjunction of ground atoms
  - First construct min subset of network necessary to answer query (generalization of KBMC)
  - Then apply MCMC (or other)
- Can also do lifted inference [Braz et al, 2005]

## Ground Network Construction



```
network ← ∅  
queue ← query nodes  
repeat  
  node ← front(queue)  
  remove node from queue  
  add node to network  
  if node not in evidence then  
    add neighbors(node) to queue  
until queue = ∅
```

## But ... Insufficient for Logic



- **Problem:**  
Deterministic dependencies break MCMC  
Near-deterministic ones make it **very** slow
- **Solution:**  
Combine MCMC and WalkSAT  
→ MC-SAT algorithm [Poon & Domingos, 2006]

## Learning



- 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



## 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, bottom-up  
[Kok & Domingos, 2005; Mihalkova & Mooney, 2007]


## 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](http://alchemy.cs.washington.edu)



	Alchemy	Prolog	BUGS
<b>Representation</b>	F.O. Logic + Markov nets	Horn clauses	Bayes nets
<b>Inference</b>	Model checking, MC-SAT	Theorem proving	Gibbs sampling
<b>Learning</b>	Parameters & structure	No	Params.
<b>Uncertainty</b>	Yes	No	Yes
<b>Relational</b>	Yes	Yes	No

## Overview



- Motivation
- Foundational areas
  - Probabilistic inference
  - Statistical learning
  - Logical inference
  - Inductive logic programming
- Putting the pieces together
- **Applications**

## Applications



- Basics
- Logistic regression
- Hypertext classification
- Information retrieval
- Entity resolution
- Hidden Markov models
- Information extraction
- Statistical parsing
- Semantic processing
- Bayesian networks
- Relational models
- Practical tips

## Running Alchemy



- Programs
  - Infer
  - Learnwts
  - Learnstruct
- Options
- MLN file
  - Types (optional)
  - Predicates
  - Formulas
- Database files

## Uniform Distribn.: Empty MLN



**Example:** Unbiased coin flips

**Type:** `flip = { 1, ... , 20 }`

**Predicate:** `Heads (flip)`

$$P(\text{Heads}(f)) = \frac{\frac{1}{Z} e^0}{\frac{1}{Z} e^0 + \frac{1}{Z} e^0} = \frac{1}{2}$$

## Binomial Distribn.: Unit Clause



**Example:** Biased coin flips

**Type:** `flip = { 1, ... , 20 }`

**Predicate:** `Heads (flip)`

**Formula:** `Heads (f)`

**Weight:** Log odds of heads:  $w = \log\left(\frac{p}{1-p}\right)$

$$P(\text{Heads}(f)) = \frac{\frac{1}{Z} e^w}{\frac{1}{Z} e^w + \frac{1}{Z} e^0} = \frac{1}{1 + e^{-w}} = p$$

By default, MLN includes unit clauses for all predicates  
(captures marginal distributions, etc.)

## Multinomial Distribution



**Example:** Throwing die

**Types:**    `throw = { 1, ... , 20 }`  
              `face = { 1, ... , 6 }`

**Predicate:** `Outcome(throw, face)`

**Formulas:** `Outcome(t, f) ^ f != f' => !Outcome(t, f')`.  
              `Exist f Outcome(t, f)`.

Too cumbersome!

## Multinomial Distrib.: ! Notation



**Example:** Throwing die

**Types:**    `throw = { 1, ... , 20 }`  
              `face = { 1, ... , 6 }`

**Predicate:** `Outcome(throw, face!)`

**Formulas:**

**Semantics:** Arguments without “!” determine arguments with “!”.  
Also makes inference more efficient (triggers blocking).

## Multinomial Distrib.: + Notation



**Example:** Throwing biased die

**Types:** throw = { 1, ... , 20 }

face = { 1, ... , 6 }

**Predicate:** Outcome(throw, face!)

**Formulas:** Outcome(t, +f)

**Semantics:** Learn weight for each grounding of args with "+".

## Logistic Regression



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

**Type:** obj = { 1, ... , n }

**Query predicate:** C(obj)

**Evidence predicates:**  $F_i(\text{obj})$

**Formulas:** a C(x)

$b_i F_i(\mathbf{x}) \wedge C(\mathbf{x})$

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

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

**Alternative form:**  $F_i(\mathbf{x}) \Rightarrow C(\mathbf{x})$

## Text Classification



```
page = { 1, ... , n }  
word = { ... }  
topic = { ... }
```

```
Topic(page, topic!)  
HasWord(page, word)
```

```
!Topic(p, t)  
HasWord(p, +w) => Topic(p, +t)
```

## Text Classification



```
Topic(page, topic!)  
HasWord(page, word)
```

```
HasWord(p, +w) => Topic(p, +t)
```

## Hypertext Classification



`Topic (page , topic!)`

`HasWord (page , word)`

`Links (page , page)`

`HasWord (p , +w) => Topic (p , +t)`

`Topic (p , t) ^ Links (p , p') => Topic (p' , t)`

**Cf.** S. Chakrabarti, B. Dom & P. Indyk, "Hypertext Classification Using Hyperlinks," in *Proc. SIGMOD-1998*.

## Information Retrieval



`InQuery (word)`

`HasWord (page , word)`

`Relevant (page)`

`InQuery (w+) ^ HasWord (p , +w) => Relevant (p)`

`Relevant (p) ^ Links (p , p') => Relevant (p')`

**Cf.** L. Page, S. Brin, R. Motwani & T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," Tech. Rept., Stanford University, 1998.

## Entity Resolution



**Problem:** Given database, find duplicate records

```
HasToken(token, field, record)
SameField(field, record, record)
SameRecord(record, record)
```

```
HasToken(+t,+f,r) ^ HasToken(+t,+f,r')
=> SameField(f,r,r')
SameField(f,r,r') => SameRecord(r,r')
SameRecord(r,r') ^ SameRecord(r',r'')
=> SameRecord(r,r'')
```

**Cf.** A. McCallum & B. Wellner, "Conditional Models of Identity Uncertainty with Application to Noun Coreference," in *Adv. NIPS 17*, 2005.

## Entity Resolution



**Can also resolve fields:**

```
HasToken(token, field, record)
SameField(field, record, record)
SameRecord(record, record)
```

```
HasToken(+t,+f,r) ^ HasToken(+t,+f,r')
=> SameField(f,r,r')
SameField(f,r,r') <=> SameRecord(r,r')
SameRecord(r,r') ^ SameRecord(r',r'')
=> SameRecord(r,r'')
SameField(f,r,r') ^ SameField(f,r',r'')
=> SameField(f,r,r'')
```

**More:** P. Singla & P. Domingos, "Entity Resolution with Markov Logic", in *Proc. ICDM-2006*.

## Hidden Markov Models



```
obs = { Obs1, ... , ObsN }  
state = { St1, ... , StM }  
time = { 0, ... , T }
```

```
State(state!, time)  
Obs(obs!, time)
```

```
State(+s, 0)  
State(+s, t) => State(+s', t+1)  
Obs(+o, t) => State(+s, t)
```

## Information Extraction



- **Problem:** Extract database from text or semi-structured sources
- **Example:** Extract database of publications from citation list(s) (the “CiteSeer problem”)
- Two steps:
  - **Segmentation:**  
Use HMM to assign tokens to fields
  - **Entity resolution:**  
Use logistic regression and transitivity

## Information Extraction



```
Token(token, position, citation)
InField(position, field, citation)
SameField(field, citation, citation)
SameCit(citation, citation)
```

```
Token(+t,i,c) => InField(i,+f,c)
InField(i,+f,c) <=> InField(i+1,+f,c)
f != f' => (!InField(i,+f,c) v !InField(i,+f',c))
```

```
Token(+t,i,c) ^ InField(i,+f,c) ^ Token(+t,i',c')
  ^ InField(i',+f,c') => SameField(+f,c,c')
SameField(+f,c,c') <=> SameCit(c,c')
SameField(f,c,c') ^ SameField(f,c',c'') => SameField(f,c,c'')
SameCit(c,c') ^ SameCit(c',c'') => SameCit(c,c'')
```

## Information Extraction



```
Token(token, position, citation)
InField(position, field, citation)
SameField(field, citation, citation)
SameCit(citation, citation)
```

```
Token(+t,i,c) => InField(i,+f,c)
InField(i,+f,c) ^ !Token("\.",i,c) <=> InField(i+1,+f,c)
f != f' => (!InField(i,+f,c) v !InField(i,+f',c))
```

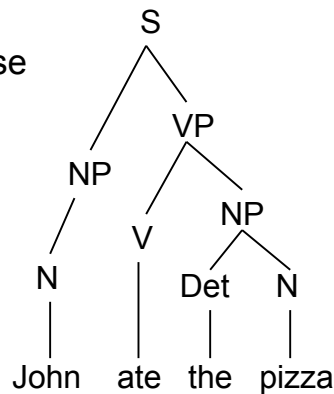
```
Token(+t,i,c) ^ InField(i,+f,c) ^ Token(+t,i',c')
  ^ InField(i',+f,c') => SameField(+f,c,c')
SameField(+f,c,c') <=> SameCit(c,c')
SameField(f,c,c') ^ SameField(f,c',c'') => SameField(f,c,c'')
SameCit(c,c') ^ SameCit(c',c'') => SameCit(c,c'')
```

**More:** H. Poon & P. Domingos, "Joint Inference in Information Extraction", in *Proc. AAAI-2007*.

## Statistical Parsing



- **Input:** Sentence
- **Output:** Most probable parse
- **PCFG:** Production rules with probabilities  
E.g.: 0.7  $NP \rightarrow N$   
0.3  $NP \rightarrow Det N$
- **WCFG:** Production rules with weights (equivalent)
- Chomsky normal form:  
 $A \rightarrow BC$  or  $A \rightarrow a$



## Statistical Parsing



- **Evidence predicate:**  $\text{Token}(\text{token}, \text{position})$   
E.g.:  $\text{Token}(\text{"pizza"}, 3)$
- **Query predicates:**  $\text{Constituent}(\text{position}, \text{position})$   
E.g.:  $\text{NP}(2, 4)$
- For each rule of the form  $A \rightarrow BC$ :  
Clause of the form  $B(i, j) \wedge C(j, k) \Rightarrow A(i, k)$   
E.g.:  $\text{NP}(i, j) \wedge \text{VP}(j, k) \Rightarrow \text{S}(i, k)$
- For each rule of the form  $A \rightarrow a$ :  
Clause of the form  $\text{Token}(a, i) \Rightarrow A(i, i+1)$   
E.g.:  $\text{Token}(\text{"pizza"}, i) \Rightarrow \text{N}(i, i+1)$
- For each nonterminal:  
Hard formula stating that exactly one production holds
- MAP inference yields most probable parse

## Semantic Processing



**Example:** John ate pizza.

**Grammar:**  $S \rightarrow NP VP$      $VP \rightarrow V NP$      $V \rightarrow \text{ate}$   
 $NP \rightarrow \text{John}$      $NP \rightarrow \text{pizza}$

$\text{Token}(\text{"John"}, 0) \Rightarrow \text{Participant}(\text{John}, E, 0, 1)$   
 $\text{Token}(\text{"ate"}, 1) \Rightarrow \text{Event}(\text{Eating}, E, 1, 2)$   
 $\text{Token}(\text{"pizza"}, 2) \Rightarrow \text{Participant}(\text{pizza}, E, 2, 3)$   
 $\text{Event}(\text{Eating}, e, i, j) \wedge \text{Participant}(p, e, j, k)$   
     $\wedge \text{VP}(i, k) \wedge V(i, j) \wedge \text{NP}(j, k) \Rightarrow \text{Eaten}(p, e)$   
 $\text{Event}(\text{Eating}, e, j, k) \wedge \text{Participant}(p, e, i, j)$   
     $\wedge S(i, k) \wedge \text{NP}(i, j) \wedge \text{VP}(j, k) \Rightarrow \text{Eater}(p, e)$   
 $\text{Event}(t, e, i, k) \Rightarrow \text{Isa}(e, t)$

**Result:**  $\text{Isa}(E, \text{Eating})$  ,  $\text{Eater}(\text{John}, E)$  ,  $\text{Eaten}(\text{pizza}, E)$

## Bayesian Networks



- Use all binary predicates with same first argument (the object  $x$ ).
- One predicate for each variable  $A$ :  $\mathbf{A}(x, \mathbf{v}!)$
- One clause for each line in the CPT and value of the variable
- Context-specific independence:  
One Horn clause for each path in the decision tree
- Logistic regression: As before
- Noisy-OR: Via logistic regression

## Relational Models



- **Knowledge-based model construction**
  - Allow only Horn clauses
  - Same as Bayes nets, except arbitrary relations
  - Combin. function: Logistic regression, noisy-OR or external
- **Stochastic logic programs**
  - Allow only Horn clauses
  - Weight of clause =  $\log(p)$
  - Add formulas: Head holds  $\Rightarrow$  Exactly one body holds
- **Probabilistic relational models**
  - Allow only binary relations
  - Same as Bayes nets, except first argument can vary

## Relational Models



- **Relational Markov networks**
  - SQL  $\rightarrow$  Datalog  $\rightarrow$  First-order logic
  - One clause for each state of a clique
  - \* syntax in Alchemy facilitates this
- **Bayesian logic**
  - Object = Cluster of similar/related observations
  - Observation constants + Object constants
  - Predicate `InstanceOf (Obs, Obj)` and clauses using it
- **Unknown relations: Second-order Markov logic**

S. Kok & P. Domingos, "Statistical Predicate Invention", in *Proc. ICML-2007*.



## Practical Tips

- Add all unit clauses (the default)
- Implications vs. conjunctions
- Open/closed world assumptions
- How to handle uncertain data:  
 $R(\mathbf{x}, \mathbf{y}) \Rightarrow R'(\mathbf{x}, \mathbf{y})$  (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



## Summary

- Most domains have multiple relations and dependencies between objects
- Much progress in recent years
- Multi-relational data mining mature enough to be practical tool
- Many old and new research issues
- Check out the Alchemy Web site:  
**[alchemy.cs.washington.edu](http://alchemy.cs.washington.edu)**