Incorporating Prior Knowledge into NLP with Markov Logic

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

Joint work with Stanley Kok, Daniel Lowd, Hoifung Poon, Matt Richardson, Parag Singla, Marc Sumner, and Jue Wang
Overview

- Motivation
- Background
- Markov logic
- Inference
- Learning
- Applications
- Discussion
Language and Knowledge: The Chicken and the Egg

- Understanding language requires knowledge
- Acquiring knowledge requires language
- “Chicken and egg” problem
- Solution: Bootstrap
  - Start with small, hand-coded knowledge base
  - Use it to help process some simple text
  - Add extracted knowledge to KB
  - Use it to process some more/harder text
  - Keep going
What’s Needed for This to Work?

- Knowledge will be very noisy and incomplete
- Inference must allow for this
- NLP must be “opened up” to knowledge
- Need joint inference between NLP and KB
- Need common representation language
  - (At least) as rich as first-order logic
  - Probabilistic
- Need learning and inference algorithms for it
- This talk: Markov logic
Markov Logic

- **Syntax:** Weighted first-order formulas
- **Semantics:** Templates for Markov nets
- **Inference:** Lifted belief propagation
- **Learning:**
  - **Weights:** Voted perceptron
  - **Formulas:** Inductive logic programming
- **Applications:** Information extraction, etc.
Overview

- Motivation
- **Background**
- Markov logic
- Inference
- Learning
- Applications
- Discussion
Markov Networks

- **Undirected** graphical models

- Potential functions defined over cliques

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

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

<table>
<thead>
<tr>
<th>Smoking</th>
<th>Cancer</th>
<th>( \Phi(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

![Graphical Model Diagram]

- Log-linear model:

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

*Weight of Feature $i$*  
*Feature $i$*  

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

\[
w_1 = 1.5
\]
First-Order Logic

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

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

- **World** (model, interpretation):
  Assignment of truth values to all ground predicates
Overview

- Motivation
- Background
- **Markov logic**
- Inference
- Learning
- Applications
- Discussion
Markov Logic

- 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)

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

\[ \forall x \text{ Smokes}(x) \implies \text{Cancer}(x) \]
\[ \forall x, y \text{ Friends}(x, y) \implies (\text{Smokes}(x) \iff \text{Smokes}(y)) \]
## Example: Friends & Smokers

<table>
<thead>
<tr>
<th>1.5</th>
<th>( \forall x \text{ Smokes}(x) \implies \text{Cancer}(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>( \forall x, y \text{ Friends}(x, y) \implies (\text{Smokes}(x) \iff \text{Smokes}(y)) )</td>
</tr>
</tbody>
</table>
Example: Friends & Smokers

1.5 $\forall x \ Smokes(x) \Rightarrow Cancer(x)$
1.1 $\forall x, y \ Friends(x, y) \Rightarrow (Smokes(x) \Leftrightarrow Smokes(y))$

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

| 1.5 | $\forall x \ Smokes(x) \Rightarrow Cancer(x)$ |
| 1.1 | $\forall x, y \ Friends(x, y) \Rightarrow (\text{Smokes}(x) \Leftrightarrow \text{Smokes}(y))$ |

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

1.5 \( \forall x \ Smokes(x) \Rightarrow Cancer(x) \)
1.1 \( \forall x, y \ Friends(x, y) \Rightarrow (Smokes(x) \Leftrightarrow Smokes(y)) \)

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

1.5 \( \forall x \ Smokes(x) \Rightarrow Cancer(x) \)

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

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

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 $x$

- 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
Overview

- Motivation
- Background
- Markov logic
- **Inference**
- Learning
- Applications
- Discussion
Belief Propagation

- Goal: Compute probabilities or MAP state
- Belief propagation: Subsumes Viterbi, etc.
- Bipartite network
  - Variables = Ground atoms
  - Features = Ground formulas
- Repeat until convergence:
  - Nodes send messages to their features
  - Features send messages to their variables
- Messages = Approximate marginals
Belief Propagation

Atoms
(x)

Formulas (f)

\[ Smokes(Ana) \]

\[ Smokes(Ana) \land Friends(Ana, Bob) \Rightarrow Smokes(Bob) \]
Belief Propagation

\[ \mu_{x \rightarrow f}(x) = \prod_{h \in n(x) \setminus \{f\}} \mu_{h \rightarrow x}(x) \]
Belief Propagation

\[ \mu_{x\rightarrow f}(x) = \prod_{h\in n(x)\setminus\{f\}} \mu_{h\rightarrow x}(x) \]

\[ \mu_{f\rightarrow x}(x) = \sum_{\sim\{x\}} \left( e^{w_f(x)} \prod_{y\in n(f)\setminus\{x\}} \mu_{y\rightarrow f}(y) \right) \]
But This Is Too Slow

- One message for each atom/formula pair
- Can easily have billions of formulas
- Too many messages!
- Group atoms/formulas which pass same message (as in resolution)
- One message for each pair of clusters
- Greatly reduces the size of the network
Belief Propagation

\[ \mu_{x \rightarrow f}(x) = \prod_{h \in n(x) \setminus \{f\}} \mu_{h \rightarrow x}(x) \]

\[ \mu_{f \rightarrow x}(x) = \sum_{\sim \{x\}} \left( e^{w_f(x)} \prod_{y \in n(f) \setminus \{x\}} \mu_{y \rightarrow f}(y) \right) \]
Lifted Belief Propagation

$$\mu_{x \rightarrow f}(x) = \prod_{h \in n(x) \setminus \{f\}} \mu_{h \rightarrow x}(x)$$

$$\mu_{f \rightarrow x}(x) = \sum_{\sim \{x\}} \left( e^{w_f(x)} \prod_{y \in n(f) \setminus \{x\}} \mu_{y \rightarrow f}(y) \right)$$
Lifted Belief Propagation

\[
\mu_{x \rightarrow f}(x) = \beta \prod_{h \in n(x) \setminus \{f\}} \mu_{h \rightarrow x}(x)
\]

\[
\mu_{f \rightarrow x}(x) = \sum_{\sim \{x\}} \left( e^{w_f(x)} \prod_{y \in n(f) \setminus \{x\}} \mu_{y \rightarrow f}(y) \right)
\]
Lifted Belief Propagation

- Form **lifted network**
  - **Supernode**: Set of ground atoms that all send and receive same messages throughout BP
  - **Superfeature**: Set of ground clauses that all send and receive same messages throughout BP

- Run belief propagation on lifted network
- Same results as ground BP
- Time and memory savings can be huge
Forming the Lifted Network

1. Form initial supernodes
   One per predicate and truth value
   (true, false, unknown)

2. Form superfeatures by doing joins of their supernodes

3. Form supernodes by projecting superfeatures down to their predicates
   Supernode = Groundings of a predicate with same number of projections from each superfeature

4. Repeat until convergence
Overview

- Motivation
- Background
- Markov logic
- Inference
- Learning
- Software
- Applications
- Discussion
Learning

- Data is a relational database
- Closed world assumption (if not: EM)
- Learning parameters (weights):
  Voted perceptron
- Learning structure (formulas):
  Inductive logic programming
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
Voted Perceptron

- Originally proposed for training HMMs discriminatively [Collins, 2002]
- Assumes network is linear chain

\[
\begin{align*}
  w_i &\leftarrow 0 \\
  \text{for } t &\leftarrow 1 \text{ to } T \text{ do} \\
  y_{MAP} &\leftarrow \text{Viterbi}(x) \\
  w_i &\leftarrow w_i + \eta \left[ \text{count}_i(y_{Data}) - \text{count}_i(y_{MAP}) \right] \\
  \text{return } \frac{\sum_t w_i}{T}
\end{align*}
\]
Voted Perceptron for MLNs

- HMMs are special case of MLNs
- Replace Viterbi by lifted BP
- Network can now be arbitrary graph

\[
\begin{align*}
    w_i &\leftarrow 0 \\
    \text{for } t &\leftarrow 1 \text{ to } T \text{ do} \\
    y_{MLN} &\leftarrow \text{LiftedBP}(x) \\
    w_i &\leftarrow w_i + \eta \left[ \text{count}_i(y_{Data}) - \text{count}_i(y_{MLN}) \right] \\
    \text{return } &\sum_t w_i / T
\end{align*}
\]
Overview

- Motivation
- Background
- Markov logic
- Inference
- Learning
- Software
- Applications
- Discussion
Parag Singla and Pedro Domingos, “Memory-Efficient Inference in Relational Domains” (AAAI-06).


Parag Singla and Pedro Domingos, “Memory-Efficient Inference in Relational Domains” (AAAI-06).


Entity Resolution

Parag Singla and Pedro Domingos, “Memory-Efficient Inference in Relational Domains” (AAAI-06).


Entity Resolution

Parag Singla and Pedro Domingos. “Memory-Efficient Inference in Relational Domains” (AAAI-06).


State of the Art

- Segmentation
  - HMM (or CRF) to assign each token to a field
- Entity resolution
  - Logistic regression to predict same field/citation
  - Transitive closure
- Markov logic implementation: Seven formulas
Types and Predicates

token = \{Parag, Singla, and, Pedro, \ldots\}
field = \{Author, Title, Venue\}
citation = \{C1, C2, \ldots\}
position = \{0, 1, 2, \ldots\}

Token(token, position, citation)
InField(position, field, citation)
SameField(field, citation, citation)
SameCit(citation, citation)
Types and Predicates

\[
\begin{align*}
\text{token} &= \{\text{Parag, Singla, and, Pedro, ...}\} \\
\text{field} &= \{\text{Author, Title, Venue, ...}\} \\
\text{citation} &= \{\text{C1, C2, ...}\} \\
\text{position} &= \{0, 1, 2, ...\}
\end{align*}
\]

Token(token, position, citation) 
InField(position, field, citation) 
SameField(field, citation, citation) 
SameCit(citation, citation)
Types and Predicates

token = \{Parag, Singla, and, Pedro, \ldots\}
field = \{Author, Title, Venue\}
citation = \{C1, C2, \ldots\}
position = \{0, 1, 2, \ldots\}

\textbf{Token}(token, position, citation) \quad \textbf{Evidence}

\textbf{InField}(position, field, citation)
\textbf{SameField}(field, citation, citation)
\textbf{SameCit}(citation, citation)
Types and Predicates

token = {Parag, Singla, and, Pedro, ...}
field = {Author, Title, Venue}
citation = {C1, C2, ...}
position = {0, 1, 2, ...}

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

← Query
Formulas

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”)
Formulas

\[
\begin{align*}
\text{Token}(+t,i,c) & \Rightarrow \text{InField}(i,+f,c) \\
\text{InField}(i,+f,c) & \Leftrightarrow \text{InField}(i+1,+f,c) \\
f \neq f' & \Rightarrow (\neg\text{InField}(i,+f,c) \lor \neg\text{InField}(i,+f',c)) \\
\text{Token}(+t,i,c) & \land \text{InField}(i,+f,c) \land \text{Token}(+t,i',c') \\
& \land \text{InField}(i',+f,c') \Rightarrow \text{SameField}(+f,c,c') \\
\text{SameField}(+f,c,c') & \Leftrightarrow \text{SameCit}(c,c') \\
\text{SameField}(+f,c,c') & \land \text{SameField}(f,c',c'') \Rightarrow \text{SameField}(f,c,c'') \\
\text{SameCit}(c,c') & \land \text{SameCit}(c',c'') \Rightarrow \text{SameCit}(c,c'')
\end{align*}
\]
Formulas

\[
\text{Token}(+t,i,c) \Rightarrow \text{InField}(i,+f,c) \\
\text{InField}(i,+f,c) \iff \text{InField}(i+1,+f,c) \\
f \neq f' \Rightarrow (!\text{InField}(i,+f,c) \lor !\text{InField}(i,+f',c))
\]

\[
\text{Token}(+t,i,c) \land \text{InField}(i,+f,c) \land \text{Token}(+t,i',c') \\
\land \text{InField}(i',+f,c') \Rightarrow \text{SameField}(+f,c,c') \\
\text{SameField}(+f,c,c') \iff \text{SameCit}(c,c') \\
\text{SameField}(f,c,c') \land \text{SameField}(f,c',c'') \\
\Rightarrow \text{SameField}(f,c,c'') \\
\text{SameCit}(c,c') \land \text{SameCit}(c',c'') \Rightarrow \text{SameCit}(c,c'')
\]
Formulas

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”)


Formulas

\( \text{Token}(+t,i,c) \implies \text{InField}(i,+f,c) \)
\( \text{InField}(i,+f,c) \iff \text{InField}(i+1,+f,c) \)
\( f \neq f' \implies ( \neg \text{InField}(i,+f,c) \lor \neg \text{InField}(i,+f',c)) \)

\[
\begin{align*}
\text{Token}(+t,i,c) \land \text{InField}(i,+f,c) \land \text{Token}(+t,i',c') \\
\land \text{InField}(i',+f,c') &\implies \text{SameField}(+f,c,c') \\
\text{SameField}(+f,c,c') &\iff \text{SameCit}(c,c') \\
\text{SameField}(f,c,c') \land \text{SameField}(f,c',c'') &\implies \text{SameField}(f,c,c'') \\
\text{SameCit}(c,c') \land \text{SameCit}(c',c'') &\implies \text{SameCit}(c,c'')
\end{align*}
\]
Formulas

$\text{Token}(+t,i,c) \Rightarrow \text{InField}(i,+f,c)$
$\text{InField}(i,+f,c) \iff \text{InField}(i+1,+f,c)$
$f \neq f' \Rightarrow (\neg \text{InField}(i,+f,c) \lor \neg \text{InField}(i,+f',c))$

$\text{Token}(+t,i,c) \land \text{InField}(i,+f,c) \land \text{Token}(+t,i',c')$
$\land \text{InField}(i',+f,c') \Rightarrow \text{SameField}(+f,c,c')$

$\text{SameField}(+f,c,c') \iff \text{SameCit}(c,c')$

$\text{SameField}(f,c,c') \land \text{SameField}(f,c',c'')$
$\Rightarrow \text{SameField}(f,c,c'')$
$\text{SameCit}(c,c') \land \text{SameCit}(c',c'') \Rightarrow \text{SameCit}(c,c'')$
Formulas

\[
\text{Token}(+t,i,c) \Rightarrow \text{InField}(i,+f,c)
\]
\[
\text{InField}(i,+f,c) \iff \text{InField}(i+1,+f,c)
\]
\[
f \neq f' \Rightarrow (\neg \text{InField}(i,+f,c) \lor \neg \text{InField}(i,+f',c))
\]

\[
\text{Token}(+t,i,c) \land \text{InField}(i,+f,c) \land \text{Token}(+t,i',c')
\]
\[
\land \text{InField}(i',+f,c') \Rightarrow \text{SameField}(+f,c,c')
\]
\[
\text{SameField}(+f,c,c') \iff \text{SameCit}(c,c')
\]
\[
\text{SameField}(f,c,c') \land \text{SameField}(f,c',c'')
\]
\[
\Rightarrow \text{SameField}(f,c,c'')
\]
\[
\text{SameCit}(c,c') \land \text{SameCit}(c',c'') \Rightarrow \text{SameCit}(c,c'')
\]
Formulas

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'')
Results: Segmentation on Cora

The graph shows the precision-recall curves for different segmentation methods. The methods compared are:
- Tokens
- Tokens + Sequence
- Tok. + Seq. + Period
- Tok. + Seq. + P. + Comma

The x-axis represents Recall, ranging from 0 to 1, and the y-axis represents Precision, also ranging from 0 to 1.
Results: Matching Venues on Cora

![Graph showing precision and recall for different similarity measures.]

- Similarity
- Sim. + Relations
- Sim. + Transitivity
- Sim. + Rel. + Trans.
Other Examples

- **Citation matching**
  Best results to date on CiteSeer and Cora
  [Poon & Domingos, AAAI-07]

- **Unsupervised coreference resolution**
  Best results to date on MUC and ACE
  (better than supervised)
  [under review]

- **Ontology induction**
  From TextRunner output (2 million tuples)
  [Kok & Domingos, ECML-08]
Overview

- Motivation
- Background
- Markov logic
- Inference
- Learning
- Applications
- Discussion
Next Steps

- Induce knowledge over ontology using structure learning
- Apply Markov logic to other NLP tasks
- Connect the pieces
- Close the loop
Conclusion

- Language and knowledge: Chicken and egg
- Solution: Bootstrap
- Markov logic provides language & algorithms
  - Weighted first-order formulas $\rightarrow$ Markov network
  - Lifted belief propagation
  - Voted perceptron
- Several successes to date
- Open-source software: Alchemy
  
  alchemy.cs.washington.edu