Presented as a poster at Statistical Relational AI (StaRAI) workshop AAAI 2010

Exploiting Causal Independence in Markov Logic Networks: Combining Undirected and Directed Models

Sriraam Natarajan
Univ of Wisconsin-Madison

Tushar Khot
Univ of Wisconsin-Madison

Daniel Lowd
Univ Of Oregon

Prasad Tadepalli
Oregon State University

Kristian Kersting
Fraunhofer IAIS, Germany

Jude Shavlik
Univ of Wisconsin-Madison

Abstract

A new method is proposed for compiling causal independencies into Markov logic networks. A Markov logic network can be viewed as compactly representing a factorization of a joint probability into the multiplication of a set of factors guided by logical formulas. We present a notion of causal independence that enables one to further factorize the factors into a combination of even smaller factors and consequently obtain a finer-grain factorization of the joint probability. The causal independence lets us specify the factor in terms of weighted, directed clauses and an associative and commutative operator, such as "or", "sum" or "max", on the contribution of the variables involved in the factors, hence combining both undirected and directed knowledge.

Introduction

Traditional AI methods were based on one of the two approaches: *first-order logic*, which excels at capturing the rich relationships among many objects, or *statistical* representations, which handle uncertain environments and noisy observations. Statistical relational learning (SRL) (Getoor and Taskar 2007), an area of growing interest, seeks to unify these approaches in order to handle problems that are both complex and uncertain.

The principal attraction of SRL models is that they are more succinct than their propositional counterparts, leading to easier specification of their structure by the domain experts and faster learning of their parameters. However, different proposed models are good at expressing different kinds of knowledge, making it difficult to compare their empirical performance. The largest divide is between directed and undirected representations.

One of the primary advantages of the directed graphical models is the notion of "Independence of Causal Influence" (ICI) (Heckerman and Breese 1994; Zhang and Poole 1996) a.k.a "causal independence," i.e., there may be multiple independent causes for a target variable. Directed models can learn conditional distributions due to each of the causes separately and combine them using a (possibly stochastic) function, thus making the process of learning easier. This notion of ICI has been extended to the directed SRL models in two different ways: while PRMs (Getoor et al. 2001) use aggregators such as `max`, `min`, and `average` to combine the influences due to several parents, other formalisms such as BLPs (Kersting and De Raedt 2007) and RBNs (Jaeger 2007) use combination functions such as Noisy-OR, mean, or weighted mean to combine distributions.

One problem with the directed models is the need to keep the graph acyclic while preserving sparsity. This problem is avoided by undirected models such as Markov logic networks (MLNs) (Domingos and Lowd 2009), which are based on Markov networks. Undirected models do not consider local models (i.e, do not treat each cause as independent from others) and hence do not model the notion of ICI explicitly. Bayesian Networks with tabular CPDs (one parameter for each configuration of the parent variables) can be directly translated to MLNs by introducing one formula for each BN parameter [1] . What is less clear is how combination functions from relational models can best be represented in an MLN. We consider a subset of combination functions called *decomposable combining rules* and derive a representation of MLNs that captures these rules. The important aspect of this representation is that we do not use the "ground" Bayesian network and instead use a "lifted" representation that avoids the grounding of the clauses.

Representing combining rules using MLNs is a key step towards unifying directed and undirected SRL approaches. Such a unified view on SRL is not only of theoretical interest. It actually has many important practical implications such as more natural model specification and development of specialized, highly efficient inference and learning techniques that can be applied differently to different pieces of the model.

In this work, we make several major contributions: (1) A provable linear representation of decomposable combining functions within MLN; (2) Explicit examples of average-based and noisy combination functions. (3) A formal description of the algorithm for converting from directed models with combining rules to MLNs. (4) Empirical proof that combining rules can improve the learning of MLNs when the domain knowledge available is minimal.

We proceed as follows. After introducing the necessary

[1] `http://alchemy.cs.washington.edu/faq/index.html`

background, we derive the MLN clauses for representing decomposable combining rules and provide the clauses for two common cases of combining rules. Next, we derive a bound on the number of clauses and provide the pseudo-code for the compilation. Before the conclusion, we present empirical results in real-world and synthetic tasks.

MLNs and Directed Models

A Bayesian network (BN) compactly represents a joint probability distribution over a set of variables $X = \{X_1, \ldots, X_n\}$ as a directed, acyclic graph and a set of conditional probability distributions (CPDs). The graph contains one node for each variable, and encodes the assertion that each variable is independent of its non-descendants given its parents in the graph. These conditional independence assertions allow us to represent the joint probability distribution as the product of the conditional probability of each variable, X_i, given its parents, parents(X_i):$P(X) = \prod_i P(X_i|\text{parents}(X_i))$

A Markov network (MN) (also called a Markov random field) specifies independencies using an undirected graph. The graph encodes the assertion that each variable is independent of all others given its neighbors in the graph. This set of independencies guarantees that the probability distribution can be factored into a set of potentials functions defined over cliques in the graph. Unlike BNs, these factors are not constrained to be conditional probabilities. Instead, a potential function is allowed to take on any non-negative value. The joint probability distribution is therefore defined as follows:$P(X = x) = \frac{1}{Z} \prod_j \phi_j(\mathbf{D}_j)$, where ϕ_j is the jth potential function, $\mathbf{D}_j$ is the set of variables over which ϕ_j is defined, and Z is a normalization constant. MNs are often written as log-linear models, where the potential functions are replaced by a set of weighted features.

One of the most popular and general SRL representations is *Markov logic networks (MLNs)* (Domingos and Lowd 2009). An MLN consists of a set of formulas in first-order logic and their real-valued weights, $\{(w_i, f_i)\}$. Together with a set of constants, we can instantiate an MLN as a Markov network with a node for each ground predicate (atom) and a feature for each ground formula. All groundings of the same formula are assigned the same weight, leading to the following joint probability distribution over all atoms:$P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right)$, where $n_i(x)$ is the number of times the ith formula is satisfied by possible world x and Z is a normalization constant (as in Markov networks). Intuitively, a possible world where formula f_i is true one more time is e^{w_i} times as probable, all other things being equal.

Directed Models with Combining Rules: Our work does not assume any representation for the directed models. We merely use an abstract syntax called as First-Order Conditional Influence (FOCI) statements (Natarajan et al. 2009) to present the semantics of the directed models. Each statement has the form: *If* $\langle condition \rangle$ *then* $\langle qualitative\ influence \rangle$, where *condition* is a set of literals, each literal being a predicate symbol applied to the appropriate number of variables. The set of literals is treated as a conjunction. A $\langle qualitative\ influence \rangle$ is of the form $X_1, \ldots, X_k$ *Qinf* Y, where the X_i and Y are of the form $V.a$, and V is a variable that occurs in *condition* and a is an object attribute. Associated with each statement is a *conditional probability distribution* that specifies a probability distribution of the resultant conditioned on the influents, e.g. $P(Y|X_1, \ldots, X_k)$ for the above statement.

```
CR2{
  If {student(S), course(C), takes(T,S,C)}
    then T.grade Qinf (CR1) S.satisfaction.
  If {student(S),paper(P,S)}
    then P.quality Qinf (CR1) S.satisfaction.
}
```

The first rule specifies that the grade that a student obtains in a course influences his/her satisfaction. The CPD $P(T.\text{grade} \mid S.\text{satisfaction})$ associated with the first statement (partially) captures the quantitative relationships between the attributes. The second states that if the student has authored a paper, then its quality influences the satisfaction of the student. The distributions due to multiple instantiations of the respective rules (the different course grades or the different paper qualities) are combined using the CR1 combining rule and the distributions due to different rules using CR2 combining rule.

Note that there are two levels of combination functions - one for combining multiple instances of the same rule and the other for combining different rules. This idea of 2-level combining rules is sufficient to capture the notion of ICI in SRL models and hence we address the 2-level combining rule in this work. The use of combining rules make learning in directed SRL models easier: multiple instances of the same rule share the same CPT and hence can be treated as individual examples while learning the CPTs. Similarly, the different CPTs can be learned *independently* of each other thus exploiting the notion of causal independence. Yet another advantage of the combining rules is that they allow for richer combination of probability distributions. MLNs in their default representation use an exponentiated weighted count as an (indirect) combination function of the different clauses. To express complex functions, a straightforward method would be to construct the grounded Bayes net for each rule and then construct the equivalent Markov net. Unfortunately, this leads to an exponential number of clauses in the MLN making the twin problems of learning and inference computationally expensive. Instead we resort to a "lifted" method that avoids unrolling (grounding) all the clauses to create the MLN.

Combination Functions using MLNs

In this section, we present the equivalent MLN representation for decomposable combining functions .

Decomposable Combining Functions

Decomposable Combination Functions are combination functions that can be implemented using deterministic or stochastic functions of the corresponding values of random variables (Natarajan et al. 2009). This is to say that there exists a value-based Bayesian Network that can capture the distribution represented using the combining rule. Note

that most common combination functions used in the literature (Koller and Pfeffer 1997; Natarajan et al. 2009; Jaeger 2007; Heckerman and Breese 1994; Zhang and Poole 1996) can be represented using the above definition. Noisy combination functions such as *Noisy-Or*, *Noisy-And*, *Noisy-Existential*, average-based combination functions such as *mean*, *weighted-mean* and *context specific Independence*(CSI) (Heckerman and Breese 1994) combining rules can also be captured by this notion of decomposable combining rule. In this work, we show how multi-level combining rules (rules that combine instances of the same clause and the ones that combine the distributions due to different clauses) can be represented and learned using MLNs.

The notion of decomposability is crucial to deriving the representation of combining rules using MLNs. This allows us to consider the combining rules as multiplexers on value-based Bayesian networks. *The key idea is to view the combination function as choosing a value among several values proposed by the parents.* For instance, taking the average of distributions corresponds to choosing the target value using an uniform distribution among the values proposed by the parents. The equivalence of weighted mean is choosing a value based on the distribution given by the weights. Similarly, the noisy combination functions can be represented using value-based network.

Consider the following two FOCI statements:

$$a(X,Y) \;\; Qinf \;\; b(Y)$$
$$c(Z,Y) \;\; Qinf \;\; b(Y)$$

where $\langle a, b, c\rangle$ are predicates and $\langle X, Y, Z\rangle$ are variables. Associated with each clause is a conditional probability distribution $P(b(y)|parent(b))$ where the parent for the first statement is $a(x, y)$ and the second is $c(z, y)$. Note that there could be several possible instantiations for X and Z in the above rules. For simplicity, let us assume that the distributions due to the different instances of the same rule are combined using CR_1 and the resulting distributions due to the different rules are combined using CR_2.

Consider the value-based network presented in Figure 1. For ease of explanation, assume that there are n instantiations of each rule and k such rules (we present only two of them for brevity). In addition to the a, b and c predicates, we introduce two more types of predicates indicated using dashed nodes: *hidden* value predicates (t and tr) and *multiplexer* predicates (h and hr). Since there are two levels of combining functions, there are two different sets of multiplexers and hidden nodes represented by two different boxes in the figure. The first box corresponds to choosing a value from a single rule (given by $r(y, i)$, where i is the rule index) and in the next level the final value of the target is chosen from one among the different r-values. We now explain the multiplexers inside the same rule (the top box) and the same idea is extended for different rules (bottom box).

The hidden predicates t's can be understood as choosing a value of the target given the instantiation of the parent based on the CPD. The multiplexers (h-nodes) serve to choose one of the n t-values for the target. The idea is that if a particular h is activated, the value of the corresponding t node is chosen to be the value of the target for the current rule (i.e, $r(y, i)$ is set to be that particular t-value). Given the different values of $r(y, I)$ for all I, the final value of the target b is chosen using the next level of the multiplexer.

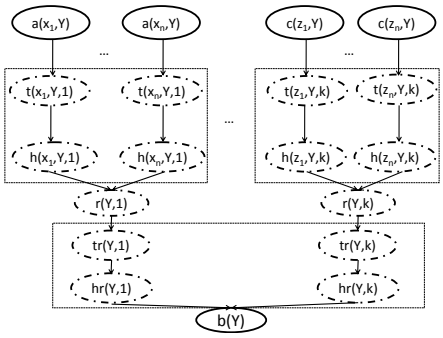

Figure 1: Value-based Bayes Net. The dashed nodes are the hidden nodes and the multiplexer nodes, while the solid nodes are observed in the data.

In our formalism, there is no restriction on the equality of CR_1 and CR_2, i.e., they need not be similar combination functions as long as they are decomposable. For instance, it is possible to use a mean combining rule to combine the instances of a single rule while a Noisy-Or could be used to combine the different rules themselves. It can be easily observed from our translation to MLNs (presented later) that the only change for the different cases would be the encoding of the multiplexers. We now present the translation of the directed models with combining rules to MLNs that consists for four different kinds of clauses:

1. **CPT Clauses:** This follows the standard translation of Bayesian Networks to MLNs. Each independent parameter in the CPT of the Bayes net becomes a clause in the MLN. An example of such a clause is

$$w_i^1 : a(X,Y) \Rightarrow t(X,Y,1)$$
$$w_i^0 : \neg a(X,Y) \Rightarrow t(X,Y,1) \qquad (1)$$

   where $w_i^j = log\frac{p_i^j}{(1-p_i^j)}$, $p_i^j = P(b(Y) = 1|a(X,Y) = j)$[2]. Hence, for each independent parameter of the original CPT in the directed model, there is a clause in the MLN with the weight as a function of the parameter.

2. **Multiplexer Clauses:** These are the clauses that choose a particular value of the target given a set of parent values. For the first-level multiplexer (h in the figure), this set corresponds to the set of values due to different instantiations of the same rule. For the second-level multiplexer, this set consists of the values due to different rules. For the first level, the MLN clauses are of the form

$$\infty : h(X,Y,I) \Rightarrow t(X,Y,I) \Leftrightarrow r(Y,I) \qquad (2)$$

   The above clause is a hard clause (i.e., infinite weight) that specifies that for a particular value of X, if $h(X, Y, i)$ is true for a rule i, then the value of the target for that rule ($r(Y, i)$) must be chosen to be the corresponding $t(X, Y, i)$. Similarly, for the next level, the multiplexer clause would be

[2] These CPT clauses are defined for rule 1 that uses predicate a. All the other rules will have similar clauses

$$\infty : hr(Y, I) \Rightarrow tr(Y, I) \Leftrightarrow b(Y) \quad (3)$$

3. **Stochastic Function Clauses:** These are the clauses that specify the stochastic function to be employed on the values. These are essentially the "prior" on the h predicates. For mean, the idea is to choose a target value from the set of h-values uniformly. In the case of Noisy-Or, the target is chosen from using an Or function over the hidden variables. We present the stochastic function clauses when we present the two different examples later in the section.
4. **Integrity Constraints:** These are the constraints that are used to specify that among the different multiplexer nodes, only **one** of them can be true for any particular example. These are of the form:

$$\begin{aligned} \infty : &\quad h(x_1, Y, I) \wedge h(x_2, Y, I) \Rightarrow (x_1 = x_2) \\ \infty : &\quad \exists X.h(X, Y, I) \end{aligned} \quad (4)$$

The above set of clauses specifies that if h is true for 2 values of X, they should be identical and there exists a grounding of X to make h-true. These constraints are exactly similar for the second level as well.

We now present two most common types of combination functions from literature: *average*-based and *Noisy* combination functions. Let us consider just a single clause $a(X, Y) \Rightarrow b(Y)$ for ease of explanation. Associated with this clause is a conditional probability distribution $P(b|a)$ (we use a and b as shorthand notations for the predicates). As we mentioned, the differences between different combination functions lie mainly in the stochastic function clauses.

Average-Based Combining Rules: Assume that the different instantiations of the above rule are combined using the weighted-mean combining rule. Then the posterior over the target b given the different sets of parents is given by

$$P(b|a_1, ..., a_n) = \frac{1}{\sum w_i} \sum_i w_i \times P(b|a_i) \quad (5)$$

where a_i denotes $a(x_i, y)$. For the case of mean, all $w_i = 1$. The CPT clauses will be of the form presented in the earlier section, where the weights are log functions of the CPT parameters ($log \frac{p_i}{(1-p_i)}$) . The multiplexer clause is again a hard clause that specifies the value of the target based on the value of the multiplexer ($h(X, Y, I)$). The integrity constraints are also the same as the ones presented above. The stochastic function is the weighted-mean. This specifies the prior on the multiplexer nodes i.e., defines the prior probability with which each multiplexer node is true. Hence, they are of the form: $u_i : h(x_i, y, i)$, where $u_i = log \frac{w_i}{\sum w_i}$ is the log-odds of the given x_i. For mean, $u_i = log(1/n)$, where n is the number of instantiations. The intuition is that each $t(X)$ chooses the value of the target based on the CPT, and the final value of the target is chosen from the different t's using the multiplexer nodes. The multiplexer is activated such that it takes only one value given by the stochastic function(mean or weighted-mean). It is easy to show mathematically that such a representation exactly captures the distribution given by equation 5. We omit the proof in the paper.

Noisy Functions: Let us assume a single rule and that the different instantiations of that rule are combined using a noisy function. For *Noisy-Or*, the marginal is computed as,

$$P(b = \mathrm{T}|a_1, \ldots, a_n) = 1 - \prod_{i=1}^{n} f_i^{a_i} \quad (6)$$

where f_i's represent the probability that a present (Boolean-valued) cause, a_i, fails to make the result b true. When converting these to MLNs, the transformation is mostly similar to the earlier case. Though the CPT clauses are constructed similarly, we present them for clarity. They are of the form:

$$\begin{aligned} \infty : &\quad \neg a(X, Y) \Rightarrow \neg t(X, Y, 1). \\ w_i : &\quad a(x_i, Y) \Rightarrow t(x_i, Y, 1). \end{aligned}$$

where, $w_i = \log((1 - f_i)/f_i)$. As can be seen, if $a(X, Y)$ is false for a particular value of X, $t(X, Y, 1)$ will always be false while if a is true, t can be false due to some noise. The multiplexer clause is similar to the earlier case, while the stochastic function (deterministic here) is given by,

$$\infty : r(Y, I) \Leftrightarrow \exists X.t(X, Y, I)$$

This asserts that $r(Y, i)$ is true if and only if some $t(X, Y, i)$ is true, which is effectively deterministic Or applied to noisy versions of the inputs. It can be shown that this set of clauses exactly capture the distribution given by equation 6.

Noisy existentials can be constructed similarly, except that we have tied weights. When constructing noisy-and, the noise adds a probability of success instead of a probability of failure:

$$\begin{aligned} w_i : &\quad \neg a(X, Y, 1) \Rightarrow \neg t(X, Y, 1) \\ \infty : &\quad a(x_i, Y, 1) \Rightarrow t(x_i, Y, 1). \end{aligned}$$

The multiplexer and the stochastic functions are also modified accordingly to reflect the *And* function.

Also, note that any MLN can be seen as a noisy-and in which the target $b(Y)$ is known to be true and each $a(x_i, Y)$ is a clause from the original MLN. Because of the infinite-weight conjunction, all a_i must be true. Since t_i is true, we can simplify each implication $\neg a(x_i, Y) \Rightarrow \neg h(x_i, Y, 1)$ to $h(x_i, Y, 1)$. The final, simplified MLN is therefore just the weighted clauses from the original MLN: $\neg w_i : \neg h(x_i, Y)$.

This formulation allows for arbitrary nesting of combining rules. The combining rules used for combining different instantiations of different rules could be different. For instance, we can imagine a situation such as $NoisyAnd(w_1\ A,\ w_2\ B,\ w_3\ NoisyOr(w_4\ C,\ w_5\ NoisyAnd(w_6\ D,\ w_7\ E,\ w_8\ F)))$ where we have both Noisy-Or and Noisy-And inside the function. w_i's are the weights while A through F are first-order logic Formulae. Such a representation is a significant generalization of MLNs.

Figure 2 describes the pseudocode for constructing MLNs from a set of FOCI statements that use combining rule CR_2. Each statement s_i has its own 1^{st} level combining rule CR_1^i. Lines 3 through 9 present the methods for constructing the clauses corresponding to s_i and its combining rule. For each independent parameter in the CPT of s_i, a clause is created. Also for each s_i, one multiplexer clause, one stochastic function and two integrity constraints are created. Once all the 1^{st} level combining rules are considered, the clauses corresponding to CR_2 are constructed in lines 10 through 14. We note that it requires $O(1)$ to construct each clause.

Fig. 2. CreateMLNClauses (FOCI Statements S, CR_2)

1: MLNClauses clauseList = [ ];
2: // Each FOCI Statement s_i has CPT, Predicates, CR_1^i
3: For Each FOCI statement $s_i \in S$
4: For Each Independent parameter θ_i^j in s_i.CPT
5: Add one CPT clause to clauseList, e.g. as in Eqn 1
6: Add to clauseList based on 1^{st} level combining rule CR_1^i:
7: One multiplexer clause as in Eqn. 2
8: One stochastic function clause, e.g. as in Eqn. 7
9: Two integrity constraint clauses as in Eqn. 4
10: For Each FOCI statement $s_i \in S$
11: Add to clauseList based on the 2^{nd} level combining rule CR_2^i:
12: One multiplexer clause as in Eqn. 3
13: One stochastic function clause, e.g. as in Eqn. 7
14: Two integrity constraint clauses as in Eqn. 4

Complexity of the resulting MLN

We now provide a bound on the number of clauses required by such an MLN. In particular, we consider the general SRL case of multi-level combining rules where the instantiations of a single rule are combined using CR_1 and different rules are combined using CR_2.

Theorem 0.1. *For any joint distribution which can be represented by* n *FOCI statements combined with nested decomposable combining rules, and* k *independent parameters, there exists an equivalent MLN of* O(nk) *rules which can be constructed in* O(nk) *time.*

Proof (sketch) The proof of equivalence is straightforward from the definition of the various clauses. Let n be the number of FOCI statements and k be the number of independent CPT parameters. From the algorithm in Figure 2, for each rule, there are k CPT clauses, one multiplexer clause, one stochastic function clause and two integrity constraints yielding $k + 4$ clauses. Hence the total number of clauses created in lines $(3 - 9)$ is $n(k + 4)$. For lines $10 - 14$ of the algorithm, the number of clauses is $n(1+1)+2 = 2(n+1)$. Hence the total number of clauses is $n(k + 6) + 2 = O(nk)$. Since each clause can be constructed in constant time given the FOCI statement and the combining rule, the resulting MLN can be constructed in $O(nk)$ time. Note that the minimal number of clauses required to model FOCI statements using MLN is $O(nk)$ as we need a clause for every parameter. Hence, our translation creates a model that is no more complex than the minimal MLN. ■

Experiments

In the following experiments, we used the Alchemy system[3] to learn the weights and/or perform inference. The same settings were used for both MLNs with combining rules (denoted by MLN^+) and the default MLNs(MLN^*). The clauses of the MLN^* are the parent configurations of the CPT of each rule. Hence, for each independent parameter of the CPT, there exists a clause in MLN^*. MLN^* was chosen so that it had the same number of parameters as that of a directed model to make a fair comparison. The clauses of MLN^+ consist of the CPT clauses and the other multiplexer, stochastic function and the integrity clauses.

Real World Datasets: In this section, we present our learning results in two real-world domains: *Cora* and *UW-CSE*. The goal of the experiment is: given minimal domain knowledge (typically 2 rules to predict the target), will the structure imposed by combining rules be useful in learning a good model? We compared MLN^* against MLN^+ for Noisy-Or combining rule. For the UW-dataset, the goal was to predict the *advisedBy* relationship between a student and a professor. The rules that we used were:

```
student(S) ∧ professor(P) ∧ course(C) ∧
taughtBy(P,C,Q) ∧ ta(S,C,Q)
⇒ advisedBy(S,P).
student(S) ∧ professor(P) ∧ publication(P,W)
∧ publication(S,W) ⇒ advisedBy(S,P).
```

MLN^* used all the combinations of the predicates in the head of the clauses and learned weights for each of them. For MLN^+, we used Noisy-Or as the combining rule at both levels. We learned the weights using Alchemy and used MC-SAT for performing inference on a test set. We present the average likelihood of the test set in the first column of Table 1. MLN^* was not able to learn reasonable weights with a small number of rules and hence predicted everything as 0. In a test-set with 50% positive examples, this yielded a likelihood of 0.5. On the other hand, with MLN^+, we were able to learn a more reasonable model that had a higher likelihood. More importantly, MLN^+ did not predict every query predicate as 0 or 1 and instead had a reasonable distribution over the target. When we added more rules to MLN^* (7 more rules from Alchemy that were earlier used in other MLN experiments to predict *advisedBy*) the average likelihood increased to 0.63. This is in line with our synthetic experiments where the performance of MLN^* improved with increasing number of rules.

Algorithm	UW	Cora
MLN^+	0.6107	0.987
MLN^*	0.5	0.963

Table 1: Results on real world domains.

The results were far more impressive in the case of *Cora* dataset where the goal is to predict whether 2 citations refer to the same one. The two rules that we used were:

```
Author(bc1,a1) ∧ Author(bc2,a2) ∧
SameAuthor(a1,a2) ⇒ SameBib(bc1,bc2).
Title(bc1,t1) ∧ Title(bc2,t2) ∧
SameTitle(t1,t2) ⇒ SameBib(bc1,bc2).
```

As can be seen from the table, MLN^+ learned nearly the perfect model for the domain and had a very high likelihood. This clearly showed that with just two rules, given some more knowledge (as hard constraints of the combining rules), MLN^+ was able to learn a highly predictive model. While MLN^* with exactly the same setting as MLN^+ predicted all the test examples as 0. We changed the settings (to generative learning, dropped some seemingly irrelevant clauses that had a large number of groundings). With these changes, we were able to get MLN^* to perform comparably with MLN^+. Admittedly,

[3] http://alchemy.cs.washington.edu/

the presence of hidden predicates increased the running time of Alchemy[4], but this motivates the need for learning algorithms that exploit the special structure efficiently (as we used the default EM learning algorithm of Alchemy to learn weights for MLN^*).

Synthetic Datasets: The goal here is to verify the accuracy of the model and understand the behavior of the combining rule clauses in the presence of varying number of objects in the domain. To this effect, we created a set of data sets with different number of rules and with different number of possible instantiations of each rule. The datasets were created using Noisy-Or and weighted mean combining rules with 1000 examples each. For each example, we chose the instantiations of the non-target predicates according to a prior, computed the marginal distribution of the target based on these instantiations and sampled the target according to the marginal. We measured the mean absolute difference between the probability values of the target for MLN^* and MLN^+ for 10 runs of each experiment.

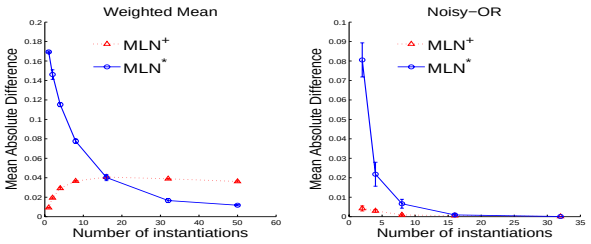

Figure 3: Results varying # of instantiations

The results corresponding to varying the number of instantiations are presented in Figure 3. The first case is the one where the first-level combining rule is mean and the second level one is weighted mean . As can be seen, for a small number of instantiations, the MLN^+ outperforms MLN^*. But as the number of instantiations increase, MLN^* is able to model the data better. This is due to the fact that as the number of instantiations (or rules) increase, the number of hidden nodes and multiplexers increase and the learning algorithms tend to introduce a small error in these cases due to the number of unobserved variables in the network.

For Noisy-Or, both the methods converge to similar estimates after about 20 instances (Figure 3). This is very much in line with the observation made in (Jaeger 1998) where the author observed that Noisy-Or is exponentially convergent. This is to say that as the number of instantiations become very large, the Noisy-Or combining rules converge to 1.0 for the target as they become a huge disjunction of all the clauses. We also varied the number of rules with constant number of instantiations and obtained very similar results for both the combining functions.

Conclusions

Combining rules capture the notion of causal independence for SRL models. We have presented the algorithm for representing a class of combining rules (decomposable combining rules) in an undirected model (MLN). We derived the equivalent clauses and provided a bound on the number of clauses required for the representation. Our experiments demonstrated that for a small number of clauses , combining functions are useful in learning more accurate models. The additional structure imposed by these functions help in guiding the learning algorithms towards reasonable weights.

However, this translation from combining rules to MLNs is not without its cost. We found that the inference in the resulting MLNs is 4-5 times slower than the one that does not use the combining rules. The problem is that while the declarative knowledge embedded in the combining rules can be encoded into clauses and given to MLNs, they have no effective means to exploit the causal independence for controlling the inference. To be effective, the inference engine has to essentially rediscover the hidden structure that is naturally exploited by the directed models. One possible future direction is to develop specialized inference algorithms that can detect structure in MLNs and exploit it for efficiency. A more general and important direction is to develop hybrid models that allow us to specify different parts of the model differently and combine them using a decomposable structure. This should allow the application of specialized learning algorithms inside each module, and combine the results in an efficient manner.

[4]The increase in running times was around 5 times on average

Acknowledgement

Sriraam Natarajan, Tushar Khot and Jude Shavlik gratefully acknowledge support of DARPA grant FA8750-09-C-0181. Prasad Tadepalli greatfully acknowledges the suppor of DARPA grant FA8750-09-C-0179. Kristian Kersting was supported by the Fraunhofer ATTRACT fellowship STREAM and by the Europen Commission under contract number FP7-248258-First-MM. Views and conclusions contained in this document are those of the authors and do not necessarily represent the official opinion or policies, either expressed or implied of the US government or of DARPA.

References

Domingos, P., and Lowd, D. 2009. *Markov Logic: An Interface Layer for AI*. San Rafael, CA: Morgan & Claypool.

Getoor, L., and Taskar, B. 2007. *Introduction to Statistical Relational Learning*. MIT Press.

Getoor, L.; Friedman, N.; Koller, D.; and Pfeffer, A. 2001. Learning probabilistic relational models. *Relational Data Mining, S. Dzeroski and N. Lavrac, Eds.*

Heckerman, D., and Breese, J. 1994. A new look at causal independence. In *UAI*.

Jaeger, M. 1998. Convergence results for relational Bayesian networks. In *Proceedings of LICS-98*.

Jaeger, M. 2007. Parameter learning for relational bayesian networks. In *ICML*.

Kersting, K., and De Raedt, L. 2007. Bayesian logic programming: Theory and tool. In *An Introduction to Statistical Relational Learning*.

Koller, D., and Pfeffer, A. 1997. Learning probabilities for noisy first-order rules. In *IJCAI*.

Natarajan, S.; Tadepalli, P.; Dietterich, T. G.; and Fern, A. 2009. Learning first-order probabilistic models with combining rules. *Special Issue on Probabilistic Relational Learning, AMAI*.

Zhang, N., and Poole, D. 1996. Exploiting causal independence in Bayesian network inference. *JAIR* 5:301–328.