Machine Learning manuscript No.
(will be inserted by the editor)

Gradient-based Boosting for Statistical Relational Learning: The Markov Logic Network and Missing Data Cases

Tushar Khot* · **Sriraam Natarajan+** · **Kristian Kersting#** · **Jude Shavlik***

Received: date / Accepted: date

Abstract Recent years have seen a surge of interest in Statistical Relational Learning (SRL) models that combine logic with probabilities. One prominent and highly expressive SRL model is Markov Logic Networks (MLNs), but the expressivity comes at the cost of learning complexity. Most of the current methods for learning MLN structure follow a two-step approach where first they search through the space of possible clauses (i.e. structures) and then learn weights via gradient descent for these clauses. We present a functional-gradient boosting algorithm to learn both the weights (in closed form) and the structure of the MLN simultaneously. Moreover most of the learning approaches for SRL apply the closed-world assumption, i.e., whatever is not observed is assumed to be false in the world. We attempt to open this assumption.We extend our algorithm for MLN structure learning to handle missing data by using an EM-based approach and show this algorithm can also be used to learn Relational Dependency Networks and relational policies. Our results in many domains demonstrate that our approach can effectively learn MLNs even in the presence of missing data.

Keywords Statistical Relational Learning · Markov Logic Networks · Missing data · Expectation Maximization

1 Introduction

Probabilistic graphical models such as Markov Networks and Bayesian Networks have been applied to many real-world problems such as diagnosis, forecasting, automated vision, and manufacturing control. A limitation of these models is that

∗ University of Wisconsin-Madison
E-mail: tushar, shavlik@cs.wisc.edu
·
+ Indiana University
E-mail: natarasr@indiana.edu
·
TU Dortmund University
E-mail: kristian.kersting@cs.tu-dortmund.de

they assume that the world can be described in terms of a fixed set of features. The world on the other hand, is made up of objects with different feature sets and relationships between the objects. Recently, there has been an increasing interest in addressing challenging problems that involve rich relational and noisy data. Consequently, several Statistical Relational Learning (SRL) methods (Getoor and Taskar, 2007) have been proposed that combine the expressiveness and compactness of first-order logic and the ability of probability theory to handle uncertainty.

While these models (Getoor et al, 2001; Kersting and De Raedt, 2007; Domingos and Lowd, 2009) are highly attractive due to their compactness and comprehensibility, the problem of learning these models is computationally intensive. This is particularly true for Markov Logic Networks (MLNs) (Domingos and Lowd, 2009) that extend Markov networks to relational settings by encoding the model as a set of weighted logic formulas. The task of learning the rules is an important and challenging task and has received much attention lately (Biba et al, 2008; Mihalkova and Mooney, 2007; Kok and Domingos, 2009, 2010). Most of these approaches, similar to propositional methods, generate candidate structures and pick the highest scoring structure. Unfortunately to score a candidate MLN structure, the weights for all the MLN rules need to be relearned. So the first problem with the existing approaches is the computational time needed for learning the structure due to repeated weight learning. The second problem is (as a consequence of the computational complexity) is that these methods learn very few (potentially long) rules limited to a reasonable amount of time.

The first goal of our work is to build upon our successful structure learning approach for learning Relational Dependency Networks (Natarajan et al, 2012) and present our learning approach for MLNs that learns the weights and the clauses simultaneously. Specifically, we turn the problem of learning MLNs into a series of relational regression problems by using Friedman's functional gradient boosting algorithm (Friedman, 2001). The key idea in this algorithm is to represent the target potential function as a series of regression trees learned in a stage-wise manner. Our proposed method greedily grows the structure without requiring any relearning of the weights, thereby drastically reducing the computation time. Hence the first problem with previous approaches is addressed by simultaneous weight and rule learning. Due to the computational efficiency and the fact that we learn shorter rules, our approach can learn many more rules in a fraction of time, thus addressing the second problem with previous approaches.

The second goal of this paper is to investigate the problem of learning the structure of SRL models in the presence of hidden (unobserved) data. Most structure learning approaches (including our previous work Natarajan et al 2012; Khot et al 2011) make the closed world assumption, i.e. whatever is unobserved in the world is considered to be false. Research with missing data in SRL has mainly focused on learning the parameters where algorithms based on classical EM (Dempster et al, 1977) have been developed (Natarajan et al, 2008; Jaeger, 2007; Xiang and Neville, 2008; Kameya and Sato, 2000; Gutmann et al, 2011; Bellodi and Riguzzi, 2013, 2012). There has also been some work on learning structure of SRL models from hidden data (Li and Zhou, 2007; Kersting and Raiko, 2005). These approaches, similar to Friedman's structural EM approach for Bayesian networks (Friedman, 1998), compute the sufficient statistics over the hidden states and perform a greedy hill-climbing search over the clauses. However, as with the fully observed case, these methods suffer from the dual problems of repeated search and computational cost.

Instead, we propose to exploit the ability of our approach to learn structure and weights simultaneously by developing an EM-based boosting algorithm for MLNs. One of the key features of our algorithm is that we consider the probability distribution to be a product of potentials and this allows us to extend this approach to Relational Dependency Networks (RDN) and relational policies. We also show how to adopt the standard approach of approximating the full likelihood by the MAP states (i.e., hard EM).

This paper makes several key contributions: (1) we propose and evaluate an algorithm that learns the structure and parameters of MLNs simultaneously (2) we propose and evaluate an algorithm to learn MLNs, RDNs and relational policies in presence of missing data and (3) the algorithm is based on two different successful methods – EM for learning with hidden data and functional-gradient boosting for SRL models – and hence is theoretically interesting. The present paper is a significant extension of our ICDM 2011 (Khot et al, 2011) paper. It provides the first coherent functional gradient boosting based learning for MLNs and presents an unified view of structure learning in MLNs in the presence of fully and partially observable data.

In the next section, we will give a brief introduction to Markov Logic Networks and Functional Gradient Boosting. In Section 3, we describe our approach to perform boosting on Markov Logic Networks. In Section 4, we extend this approach to handle missing data. We present the experimental results in Section 5, starting with the experiments for boosted MLNs for fully observed data, followed by the experiments for learning RDNs, MLNs and relational policies with partially observed data. Finally, we conclude with the future extensions of our work.

2 Background

We first define some notations that will be used in this work. We use capital letters such as X, Y, and Z to represent random variables (atoms in our formalism). However, when writing sentences in first-order predicate calculus, we use sans-serif capital letters X, Y, and Z to represent logical variables. All logical variables are implicitly universally quantified (i.e. $\forall$) unless we explicitly existentially quantify them. We use small letters such as x, y, and z to represent values taken by the variables (and x, y and z to represent values taken by logical variables i.e. objects in the domain). We use bold-faced letters to represents sets. Letters such as $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$ represent sets of variables and $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ represent sets of values. We use $\mathbf{z}_{-z}$ to denote $\mathbf{z} \setminus z$ and $\mathbf{x}_{-i}$ to represent $\mathbf{x} \setminus x_i$.

2.1 Markov Logic Networks

A popular SRL representation 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, $\{(w_i, f_i)\}$. Higher the weight of the rule, more likely it is to be true in the world. Together with a set of constants, we can instantiate an MLN as a Markov network with a variable node for each ground predicate and a factor for each ground formula. All factors corresponding to the groundings of the

same formula are assigned the same potential function ($\exp(w_i)$), leading to the following joint probability distribution over all atoms:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(\mathbf{x})\right) \tag{1}$$

where $n_i(\mathbf{x})$ is the number of times the ith formula is satisfied by the instantiation of the variable nodes, $\mathbf{x}$ and Z is a normalization constant (as in Markov networks). Intuitively, an instantiation where formula f_i is true one more time than a different possible instantiation is e^{w_i} times as probable, all other things being equal. For a detailed description of MLNs, we refer to the book (Domingos and Lowd, 2009).

We consider the problem of structure learning where the goal is to learn the weighted clauses given data. Bottom-up structure learning (Mihalkova and Mooney, 2007) uses a propositional Markov network learning algorithm to identify paths of ground atoms. These paths form the templates that are generalized into first-order formulas. Hypergraph lifting (Kok and Domingos, 2009) clusters the constants and true atoms to construct a lifted (first-order) graph. Relational path-finding on this hypergraph is used to obtain the MLN clauses. Structural motif learning (Kok and Domingos, 2010) uses random walks on the ground network to find symmetrical paths and cluster nodes with similar path distributions. These methods rely on variations of path finding approaches to reduce the space of possible structures. As a result if a relevant path is missed, it would not be considered as a candidate structure. Discriminative structure learner (Biba et al, 2008) performs local search over the space of first-order logic clauses using perturbation operators (add, remove, flip). The candidate structures are not constrained by any relational paths but scoring each structure is computationally intensive.

These methods obtain the candidate rules first, learn the weights and update the structure by scoring the candidate structures. For every candidate, the weights of the rules need to be learned which can not be calculated in closed-form. Consequently, most of these methods are slow and learn a small set of rules. As mentioned earlier, our algorithm learns the structure and parameters simultaneously.

2.2 Functional Gradient Boosting

Consider the standard method of supervised parameter learning using gradient-descent. The learning algorithm starts with initial parameters θ_0 and computes the gradient of the log-likelihood function ($\Delta_1 = \frac{\partial}{\partial \theta} \log P(\mathbf{X}; \theta_0)$). Friedman (2001) developed an alternate approach to perform gradient descent where the log-likelihood function is represented using a regression function $\psi : \mathbf{x} \rightarrow \mathbb{R}$ over the example space $\mathbf{x}$ (feature vectors in propositional domains) and the gradients are performed with respect to $\psi(x)$. For example, the likelihood of x being true can be represented as $P(x; \psi) = sigmoid(\psi(x))$[1] and gradients can be calculated as $\Delta(x) = \frac{\partial}{\partial \psi(x)} \sum_x^{\mathbf{x}} logP(x; \psi)$.

Similar to parametric gradients, functional gradient descent starts with an initial function ψ_0 and iteratively adds gradients Δ_m. Each gradient term (Δ_m)

[1] $sigmoid(x) = \frac{1}{1+e^{-x}}$

is a regression function over the entire example space. Since the space of examples can be very large, the gradients are computed for only the training examples. So the gradients at the m^{th} iteration can be represented as $\langle x_i, \Delta_m(x_i)\rangle$ where $x_i \in$ training examples. Also rather than directly using $\langle x_i, \Delta_m(x_i)\rangle$ as the gradient function, functional gradient boosting *generalizes* by fitting a regression function $\hat{\psi}_m$ (generally regression trees) to the gradients Δ_m (e.g. $\hat{\psi}_m = \arg\min_\psi \sum_x [\psi(x) - \Delta_m(x)]^2$). Thus, each gradient step ($\hat{\psi}_m$) is a regression tree and the final model $\psi_m = \psi_0 + \hat{\psi}_1 + \cdots + \hat{\psi}_m$ is a sum over these regression trees.

2.3 Relational Functional Gradient

Functional gradient boosting has been applied to various relational models for learning the structure (Natarajan et al, 2012; Karwath et al, 2008; Kersting and Driessens, 2008; Natarajan et al, 2011). Similar to propositional approaches, relational approaches define the probability function to be a sigmoid function over the ψ function. The examples are ground atoms of the target predicate such as `advisedBy(anna, bob)`. But since these are relational models, the ψ function depends on all the ground atoms and not just the attributes of a given example. The ψ function is represented using relational regression trees (RRT)(Blockeel and Raedt, 1998). For example, the probability function used by Natarajan et al (2012) to learn the structure of Relational Dependency Networks was: $P(x_i) = sigmoid(\psi(x_i; Pa(x_i)))$ where $Pa(x_i)$ are all the relational/first-order logic facts that are used in the RRTs learned for x_i. They showed that the functional gradient of likelihood for RDNs as $\frac{\partial P(\mathbf{X}=\mathbf{x})}{\partial \psi(X_i)} = I(x_i = 1) - P(X_i = 1; Pa(X_i))$ which is the difference between the true distribution (I is the indicator function) and the current predicted distribution. For positive examples, the gradient is always positive and pushes the ψ function value ($\psi_0 + \Delta_1 + \cdots + \Delta_m$) closer to ∞ and the probability value closer to 1. Similarly the gradients for negative examples is negative and hence the ψ function is pushed closer to $-\infty$ and probability closer to 0. Most objective functions in propositional and relational models result in this gradient function. However, due to difference in semantics of MLNs, we derive a new scoring function (particularly in the case of hidden data).

3 MLN Structure Learning with Fully Observed Data

Maximizing the likelihood of the training data is the most common approach taken for learning many probabilistic models. However, in the case of Markov networks and MLNs, it is computationally infeasible to maximize this likelihood (given in Equation 1) due to the normalization constant. Hence, we follow the popular approach of maximizing the pseudo-likelihood (PL) (Besag, 1975) given by, $PL(\mathbf{X} = \mathbf{x}) = \prod_{x_i \in \mathbf{x}} P(x_i|\mathbf{MB}(x_i))$ where $\mathbf{MB}(x_i)$ is the Markov blanket of x_i. The Markov blanket in MLNs are the neighbors of x_i in the grounded network. Recall the probability distribution of MLNs is defined as $P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(\mathbf{x})\right)$. Given this definition, the conditional probability $P(x_i|\mathbf{MB}(x_i))$ can be represented as

$$P(x_i = 1|\mathbf{MB}(x_i)) = \frac{P(x_i = 1, \mathbf{MB}(x_i))}{\sum_{x' \in \{0,1\}} P(x_i = x', \mathbf{MB}(x_i))}$$

$$
\begin{aligned}
&\text{// We assume boolean variables} \\
&= \frac{\exp\left(\sum_j w_j nt_j(x_i; \mathbf{MB}(x_i))\right)}{\exp\left(\sum_j w_j nt_j(x_i; \mathbf{MB}(x_i))\right) + 1} \qquad (2) \\
&\text{// On dividing by } \exp(\sum_j w_j n_j(x_i = 0, MB(x_i)))
\end{aligned}
$$

$$
\text{where } nt_j(x_i; \mathbf{MB}(x_i)) = n_j(x_i = 1, \mathbf{MB}(x_i)) - n_j(x_i = 0, \mathbf{MB}(x_i)) \qquad (3)
$$

$n_j(\mathbf{x})$ is the number of groundings of rule C_j given the instantiation $\mathbf{x}$. $nt_j(x_i; \mathbf{MB}(x_i))$ corresponds to the non-trivial groundings (Shavlik and Natarajan, 2009) of an example x_i given its Markov blanket (we explain non-trivial groundings later) . We define the potential function as $\psi(x_i; \mathbf{MB}(x_i)) = \sum_j w_j nt_j(x_i; \mathbf{MB}(x_i))$. As a result Equation 2 for probability of an example being true can be rewritten as

$$
\begin{aligned}
P(x_i = 1|\mathbf{MB}(x_i)) &= \frac{\exp\left(\psi(x_i; \mathbf{MB}(x_i))\right)}{\exp\left(\psi(x_i; \mathbf{MB}(x_i))\right) + 1} \qquad (4) \\
\Rightarrow P(x_i|\mathbf{MB}(x_i)) &= \frac{\exp\left(\psi(x_i; \mathbf{MB}(x_i)) \times I(x_i = 1)\right)}{\exp\left(\psi(x_i; \mathbf{MB}(x_i))\right) + 1}
\end{aligned}
$$

where $I(x_i = 1)$ returns 1, if x_i is true and returns 0 otherwise. Now,

$$
\begin{aligned}
PLL(\mathbf{X} = \mathbf{x}) &= \sum_{x_i \in \mathbf{x}} \log P(x_i|\mathbf{MB}(x_i)) \\
&= \sum_{x_i \in \mathbf{x}} \left[\psi(x_i; \mathbf{MB}(x_i)) \times I(x_i = 1) - \log\left(\exp\left(\psi(x_i; \mathbf{MB}(x_i))\right) + 1\right)\right]
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial PLL(\mathbf{X} = \mathbf{x})}{\partial \psi(x_i; \mathbf{MB}(x_i))} &= \frac{\partial \log P(x_i; \mathbf{MB}(x_i))}{\partial \psi(x_i; \mathbf{MB}(x_i))} \\
&= \frac{\partial \left[\psi(x_i; \mathbf{MB}(x_i)) \times I(x_i = 1) - \log\left(\exp\left(\psi(x_i; \mathbf{MB}(x_i))\right) + 1\right)\right]}{\partial \psi(x_i; \mathbf{MB}(x_i))} \\
&= I(x_i = 1) - \frac{\partial \log\left(\exp\left(\psi(x_i; \mathbf{MB}(x_i))\right) + 1\right)}{\partial \psi(x_i; \mathbf{MB}(x_i))} \\
&= I(x_i = 1) - \frac{\exp\left(\psi(x_i; \mathbf{MB}(x_i))\right)}{\exp\left(\psi(x_i; \mathbf{MB}(x_i))\right) + 1} \\
\Delta(x_i) &= I(x_i = 1) - P(x_i = 1; \mathbf{MB}(x_i)) \qquad (5)
\end{aligned}
$$

The gradient ($\Delta(x_i)$) at each example x_i is now simply the adjustment required for the probabilities to match the observed value for that example. This gradient serves as the value of the target function for the current regression example at the next training episode. The expression in Equation 5 is very similar to the one derived for previous functional gradient boosting methods (Dietterich et al, 2008; Kersting and Driessens, 2008; Natarajan et al, 2012; Karwath et al, 2008). The key feature of the above expression is that the functional gradient for each example is dependent on the observed value. If the example is positive, the gradient $(I - P)$ is positive indicating that the model should increase the probability of the ground predicate being true. If the example is negative, the gradient is negative, implying that it will push the probability towards 0.

We now explain the notion of non-trivial groundings ($nt_j(x_i)$) described in Equation 3. Consider the horn clause, $p_1(\mathbf{X}_1) \wedge \ldots \wedge p_c(\mathbf{X}_c) \rightarrow target(\mathbf{X}')$. We will represent the clause as $\wedge_k p_k(\mathbf{X}_k) \rightarrow target(\mathbf{X}')$ where $\mathbf{X}_k$ are the arguments for p_k and $\mathbf{X}' \subseteq \cup_k \mathbf{X}_k$. Groundings of C_j that remain true irrespective of the truth value of a ground atom x_i (say $target(\mathbf{x})$) are defined as the *trivial* groundings, while others are called as the *non-trivial* groundings. The clause $\wedge_k p_k(\mathbf{X}_k) \rightarrow target(\mathbf{X}')$, i.e. $\vee_k \neg p_k(\mathbf{X}_k) \vee target(\mathbf{X}')$ is true irrespective of the value of $target(\mathbf{X}')$ when $\vee_k \neg p_k(\mathbf{X}_k) = true$. These groundings correspond to the trivial groundings and therefore the non-trivial groundings correspond to $\vee_k \neg p_k(\mathbf{X}_k) = false$, i.e. $\wedge p_k(\mathbf{X}_k) = true$. Hence, the non-trivial groundings of the clause $\wedge_k p_k(\mathbf{X}_k) \rightarrow target(\mathbf{X}')$ would correspond to the groundings for $\cup_k \mathbf{X}_k$ that satisfy the body of the clause, i.e. $\wedge_k p_k(\mathbf{x}_k) = true$. Since we only need to consider the groundings of the clause that contain the ground atom x_i, we check for $\wedge_k p_k(\mathbf{X}_k) = true$ after unifying the head of the clause ($target(\mathbf{X}')$) with the ground atom, x_i. For example, the non-trivial groundings of $p(\mathsf{X}) \wedge q(\mathsf{X}, \mathsf{Y}) \rightarrow target(\mathsf{X})$ for the example $target(\mathsf{x}_1)$ correspond to the groundings $\{\mathsf{Y} | p(\mathsf{x}_1) \wedge q(\mathsf{x}_1, \mathsf{Y}) = true\}$. For a more detailed discussion of non-trivial groundings, see Shavlik and Natarajan (2009).

3.1 Representation of Functional Gradients for MLNs

Given the gradients for each example (Equation 5), our next goal is to find the potential function $\hat{\psi}$ such that the squared error between $\hat{\psi}$ and the functional gradient is minimized. We use two representations of $\hat{\psi}$s: *trees* and *clauses*.

3.1.1 Tree representation

Following prior work (Natarajan et al, 2012), we use a relational regression tree learner (modified from TILDE (Blockeel and Raedt, 1998)) to model the functional gradients. Hence, the final potential function corresponds to the set of RRTs learned from data. Every path from the root to a leaf is a clause and the value of the leaf corresponds to the weight of the clause. For example, the regression tree in Figure 1 can be represented with the following clauses:

$w_1 : p(\mathsf{X}) \wedge q(\mathsf{X}, \mathsf{Y}) \rightarrow target(\mathsf{X})$, $w_2 : p(\mathsf{X}) \wedge \neg \exists Y q(\mathsf{X}, \mathsf{Y}) \rightarrow target(\mathsf{X})$, and $w_3 : \neg p(\mathsf{X}) \rightarrow target(\mathsf{X})$

The MLN corresponding to these weighted clauses matches the model represented using our trees. Hence these rules can now be used with any other MLN inference engine. Note that TILDE trees (Blockeel and Raedt, 1998) employ weighted variance as the splitting criterion at every node and it ignores the number of groundings for a predicate. On the other hand MLN semantics require counting of the number of non-trivial groundings and hence we modify the splitting criterion at every node. As an example, let us consider building the tree presented in

Fig. 1: Example tree for target(X)

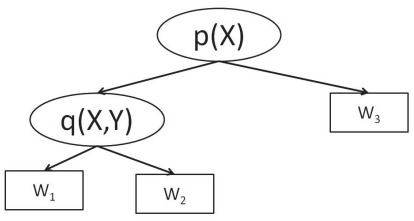


Figure 1. Assume that we already have determined $p(\mathsf{X})$ as the root and now we consider adding $q(\mathsf{X},\mathsf{Y})$ to node N (at left branch). So adding $q(\mathsf{X},\mathsf{Y})$ would result in two clauses, $C_1 : p(\mathsf{X}) \wedge q(\mathsf{X},\mathsf{Y}) \rightarrow target(\mathsf{X})$ and $C_2 : p(\mathsf{X}) \wedge \neg\exists Y\, q(\mathsf{X},\mathsf{Y}) \rightarrow target(\mathsf{X})$. For all the examples that reach the node N, assume $\mathcal{I}$ to be the set of examples that satisfy $q(\mathsf{X},\mathsf{Y})$ and $\mathcal{J}$ be the ones that do not. Let w_1 and w_2 be the regression values that would be assigned to C_1 and C_2 respectively. Let $n_{x,1}$ and $n_{x,2}$ be the number of non-trivial groundings for an example x with clauses C_1 and C_2. The regression value returned for an example would depend on whether it belongs to $\mathcal{I}$ or $\mathcal{J}$. Hence the regression value returned by this tree for x is

$$\hat{\psi}(x_i) = n_{x_i,1} \cdot w_1 \cdot I(x_i \in \mathcal{I}) + n_{x_i,2} \cdot w_2 \cdot I(x_i \in \mathcal{J}) \tag{6}$$

and the squared error is used to compute the gradients.

$$SE = \textstyle\sum_{x\in\mathcal{I}} [n_{x,1} \cdot w_1 - \Delta(x)]^2 + \sum_{x\in\mathcal{J}} [n_{x,2} \cdot w_2 - \Delta(x)]^2$$

$$\frac{\partial}{\partial w_1} SE = \textstyle\sum_{x\in\mathcal{I}} 2 \cdot [n_{x,1} \cdot w_1 - \Delta(x)] \cdot n_{x,1} + 0 = 0 \Longrightarrow w_1 = \frac{\sum_{x\in\mathcal{I}} \Delta(x) \cdot n_{x,1}}{\sum_{x\in\mathcal{I}} n_{x,1}^2}$$

$$\frac{\partial}{\partial w_2} SE = 0 + \textstyle\sum_{x\in\mathcal{J}} 2 \cdot [n_{x,2} \cdot w_2 - \Delta(x)] \cdot n_{x,2} = 0 \Longrightarrow w_2 = \frac{\sum_{x\in\mathcal{J}} \Delta(x) \cdot n_{x,2}}{\sum_{x\in\mathcal{J}} n_{x,2}^2}$$

For every potential split, we can calculate the weights using these equations and use these weights to calculate the squared error. We greedily select the literal at each node that minimizes this squared error.

Generally, the false branch at every node with condition $C(\mathbf{X})$, would be converted to $\forall \mathbf{X}_f, \neg C(\mathbf{X})$ which in its CNF form becomes $\neg\exists \mathbf{X}_f, C(\mathbf{X})$, where $\mathbf{X}_f \subset \mathbf{X}$ are the free variables in $C(\mathbf{X})$. Existentially defined variables can result in a large clique in the ground Markov Network. To avoid this, we maintain an ordered list of clauses (Blockeel and Raedt, 1998) and return the weight of the first clause that has at least one true grounding for a given example. We can then ignore the condition in a given node for the false branch to the leaf. It is worth noting that $C(\mathbf{X})$ maybe a conjunction of literals depending on the maximum number of literals allowed at an inner node. Following TILDE semantics, we can also allow aggregation functions inside the node where the evaluation will always result in a single grounding of the aggregation predicate.

Figure 1 gives an example regression tree for $target(\mathsf{X})$. If we are scoring the node $q(\mathsf{X},\mathsf{Y})$, we split all the examples that satisfy $p(\mathsf{X})$ into two sets $\mathcal{I}$ and $\mathcal{J}$. $\mathcal{I}$ contains all examples that have at least one grounding for $q(\mathsf{X},\mathsf{Y})$ and $\mathcal{J}$ contains the rest; $target(\mathsf{x}_1)$ would be in $\mathcal{I}$ if $p(\mathsf{x}_1) \wedge q(\mathsf{x}_1,\mathsf{Y})$ is true and $target(\mathsf{x}_2)$ would be in $\mathcal{J}$, if $p(\mathsf{x}_2) \wedge (\forall \mathsf{Y}, \neg q(\mathsf{x}_2,\mathsf{Y}))$ is true. The parameter $n_{\mathsf{x}_1,1}$ corresponds to the number of groundings of $p(\mathsf{x}_1) \wedge q(\mathsf{x}_1,\mathsf{Y})$, while $n_{\mathsf{x}_2,2}$ corresponds to the number of groundings of $p(\mathsf{x}_2) \wedge (\forall \mathsf{Y}, \neg q(\mathsf{X}_2,\mathsf{Y})$. The corresponding ordered list of MLN rules is: $w_1 : p(\mathsf{X}), q(\mathsf{X},\mathsf{Y}) \rightarrow target(\mathsf{X})$, $w_2 : p(\mathsf{X}) \rightarrow target(\mathsf{X})$ $w_3 : target(\mathsf{X})$.

3.1.2 Clause representation

We also learn Horn clauses directly instead of trees by using a beam search that adds literals to clauses that reduce the squared error. We maintain a (beam-size limited) list of clauses ordered by their squared error and keep expanding clauses

from the top of the list. We add clauses as long as their lengths do not exceed a threshold and the beam still contains clauses. We recommend using clauses when the negation-by-failures introduced by the trees can render inference using standard MLN software (since they can not handle ordered decision lists) computationally intensive. Essentially, we replace the tree with a set of clauses learned independently at each gradient-step. Since we only have the true branch when a new condition is added, the error function becomes:

$$SE = \sum_{x \in \mathcal{I}} [n_{x,1} \cdot w - \Delta_x]^2 + \sum_{x \in \mathcal{J}} \Delta_x^2 \Longrightarrow w = \frac{\sum_{x \in \mathcal{I}} \Delta_x \cdot n_{x,1}}{\sum_{x \in \mathcal{I}} n_{x,1}^2}$$

Note that the key change is that we do not split the nodes and instead just search for new literals to add to the current set of clauses. Hence, instead of a RRT for each gradient step, we learn a pre-set number of clauses (C). We use a similar parameter for the RRT learner as well with a maximum number of allowed leaves (L). The values of C and L are fixed at 3 and 8 respectively for all our experiments. Hence, the depth of tree is small and so is the number of clauses per gradient-step.

3.2 Algorithm for Learning MLNs

Before presenting the algorithm, we summarize the strengths of our approach. Apart from learning the structure and weight simultaneously, this approach has two other key advantages. One, the models are essentially weighted Horn clauses. This makes the inference process easier, especially given that we use the procedure presented in Shavlik and Natarajan (2009) to keep track of the non-trivial groundings for a clause/predicate. Secondly, our learning algorithms can use prior knowledge as an initial set of MLN clauses and learn more clauses as needed.

Algorithm 1 MLN-Boost: FGB for MLNs

```
1: function TreeBoostForMLNs(Data)
2:     F_0 := Initial Model
3:     for 1 ≤ m ≤ M do                                  ▷ M gradient steps
4:         F_m := F_{m-1}
5:         for P in T do                                 ▷ Iterate through target predicates
6:             S := GenExamples(Data; F_{m-1}, P)        ▷ Generate examples
7:             Δ_m := FitRelRegressionTree(S, P, L)      ▷ Fit trees to gradients
8:             F_m := F_m + Δ_m                          ▷ Update model
9:         end for
10:    end for
11:    P(x_i|MB(x_i)) ∝ F_m(x_i)                         ▷ Obtained by grounding F_M
12: end function
```

Algorithm 1 presents functional gradient boosting of MLNs with both the tree and the clause learner. In $TreeBoostForMLNs$, we iterate through M gradient steps and in each gradient step learn a regression tree for the target predicates one at a time. We create examples for the regression learner for a given predicate, P using the $GenExamples$ method. We use the function $FitRelRegressionTree(S, P, L)$

Algorithm 2 Tree Learning for MLNs

```
1: function FitRelRegressionTree(S, P, L)
2:     Tree := createTree(P(X))
3:     Beam := {root(Tree)}
4:     while numLeaves(Tree) ≤ L do
5:         Node := popBack(Beam)              ▷ Expand leaf node with worst score
6:         C := createChildren(Node)          ▷ Create possible children
7:         BN := popFront(Sort(C))            ▷ Pick node with best score
8:         addNode(Tree, Node, BN)            ▷ Replace leaf node with BN
9:         insert(Beam, BN.left, BN.left.score)
10:        insert(Beam, BN.right, BN.right.score)
11:    end while
12: return Tree
13: end function
```

Algorithm 3 Clause learning for MLNs

```
1: function FitRelRegressionClause(S, P, N, B)
2:     Beam := {P(X)}
3:     BC := P(X)
4:     while ¬empty(Beam) do
5:         Clause := popFront(Beam)           ▷ Best scoring clause
6:         if length(Clause) ≥ N then
7:             continue                       ▷ Clause cannot be expanded
8:         end if
9:         C := addLiterals(Clause)
10:        for c ∈ C do
11:            c.score = SE(c)                ▷ Squared error
12:            if c.score ≥ Clause.score then ▷ If the new clause has a better score
13:                insert(Beam, c, c.score)   ▷ Add new clause to beam
14:            end if
15:            if c.score ≥ BC.score then
16:                BC := c                    ▷ Update best clause
17:            end if
18:        end for
19:        while length(Beam) ≥ B do          ▷ Remove low scoring clauses
20:            popBack(Beam)
21:        end while
22:    end while
23: return BC
24: end function
```

to learn a tree that best fits these examples. We limit our trees to have maximum L leaves and greedily pick the best node to expand. We set $L = 8$ and $M = 20$.

FitRelRegressionClause(S, P, N, B) can be called here to learn clauses instead. N is the maximum length of the clause and B is the maximum beam size. In *FitRelRegressionTree*, we begin with an empty tree that returns a constant value. We use the background predicate definitions (mode specifications) to create the potential literals that can be added (*createChildren*). We pick the best scoring node (based on square error) and replace the current leaf node with the new node (*addNode*). Then both the left and right branch of the new node are added to the potential list of nodes to expand. To avoid overfitting, we only insert and hence expand nodes that have at least 6 examples. We pick the node with the worst score and repeat the process.

The function for learning clauses is shown as $FitRelRegressionClause$ which takes the maximum clause length as the parameter, N (we set this to 3) and beam size, B (we set this to 10). It greedily tries to find the best scoring clause (BC) with $length \leq N$. This method only learns one clause at a time. Hence for learning multiple clauses, we call this function multiple times (thrice in our experiments) during one step and update the gradients for each example before each call.

3.3 Learning Joint Models

To handle multiple target predicates, we learn a joint model by learning tree/clauses for each predicate in turn. We use the MLN's learned for all the predicates prior to the current iteration to calculate the regression value for the examples. We implement this by learning one tree for every target predicate in line 4 in Algorithm 1. For efficiency, while learning a tree for one target predicate, we do not consider the influence of that tree on other predicates.

Since we use the clauses learned for other predicates to compute the regression value for an example, we have to handle cases where the examples unify with a literal in the body of the clause. Consider the clause, $C_j : p(\mathsf{X}), q(\mathsf{X}, \mathsf{Y}) \rightarrow target(\mathsf{X})$. If we learn a joint model over $target$ and p, this clause will be used to compute the regression value for $p(\mathsf{X})$ in the next iteration. In such a case, the number of non-trivial groundings corresponding to an example, say $p(\mathsf{x})$ for a given grounding $(\mathsf{X} = \mathsf{x})$ and the given clause would be the number of groundings of $q(\mathsf{x}, \mathsf{Y}) \wedge \neg target(\mathsf{x})$. Since $p(\mathsf{x})$ appears in the body of the clause, the difference $nt_j(p(\mathsf{x})) = [n_j(p(\mathsf{x}) = 1) - n_j(p(\mathsf{x}) = 0)]$ would be negative. $nt_j(p(\mathsf{x}))$ is simply the negative of number of non-trivial groundings of $p(\mathsf{x})$ (or non-trivial groundings of $\neg p(\mathsf{x})$) for the above clause. Computing $nt_j(\mathsf{x}_i)$ this way allows us to compute the $\hat{\psi}$ values for every example quickly without grounding the entire MLN at every iteration as the number of groundings can be simply negated in some cases. We incrementally calculate the number of groundings for every clause (one literal at a time) and store the groundings at every node to prevent repeated computations.

4 Handling Missing Data

We presented our approach to learn structure for MLNs for fully observed data where we made the closed world assumption for hidden data. We now aim to relax this assumption by extending the functional gradient boosting to perform iterative learning similar to the EM algorithm. As with EM, the iterative approach has two steps the first of which computes the expected value of the hidden predicates and the second maximizes the likelihood of the data given the current expected values.

Before deriving the E and M steps, we present the high-level overview of our RFGB-EM (Relational Functional Gradient Boosting - EM) approach in Figure 2. Similar to other EM approaches, we sample the states for the hidden groundings based on our current model in the E-step and use the sampled states to update our model in the M-step. ψ_t represents the model in the t^{th} iteration. The initial model, ψ_0 can be as simple as a uniform probability for all examples or could be a model specified by an expert. We sample certain number of assignments of the hidden groundings (denoted as $|W|$) using the current model ψ_t. Based on these samples, we create regression examples which are then used to learn T relational

regression trees as presented in the previous section. The learned regression trees are added to the current model and the process is repeated. To perform the M-step, we update the current model using functional gradients.

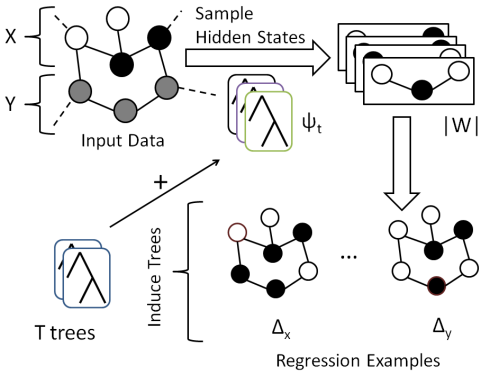


Fig. 2: RFGB-EM in action. Shaded nodes indicate variables with unknown assignments, while the white (or black) nodes are assigned true (or false) values. The input data has observed (indicated by $\mathbf{X}$) and hidden (indicated by $\mathbf{Y}$) groundings. We sample $|W|$ assignments of the hidden groundings using the current model ψ_t. We create regression examples based on these samples, which are used to learn T relational regression trees. The learned trees are added to the current model and the process is repeated.

4.1 Derivation for M-step

As mentioned before, we again use upper case letter as random variables, lower case letters as variable assignments and bold face letters for sets. For ease of explanation, let $\mathbf{X}$ be all the observed predicates and $\mathbf{Y}$ be all the hidden predicates (their corresponding groundings are $\mathbf{x}$ and $\mathbf{y}$). Since $\mathbf{Y}$ is unobserved, it can have multiple possible assignments denoted by $\mathcal{Y}$ and $\mathbf{y} \in \mathcal{Y}$ represents one such hidden state assignment. We assume that some of the groundings of the hidden predicates are always observed. Unlike parameter learning approaches where a latent predicate (with all groundings missing) would be part of the model, a structure learning approach will not even use a latent predicate in its model. We also assume that true atoms and hidden atoms are provided for the hidden predicates and the remaining ground atoms are known to be false.

Traditionally, EM approaches for parameter learning find θ that maximize the $\mathcal{Q}(\theta|\theta_t)$ function. The $\mathcal{Q}(\theta|\theta_t)$ function is defined as the expected log-likelihood of missing and observed data (based on θ) where the expectation is measured based on the current distribution (θ_t) for the missing data, i.e.

$$\mathcal{Q}(\theta|\theta_t) = \sum_{\mathbf{y} \in \mathcal{Y}} P(\mathbf{y}|\mathbf{x}; \theta_t) \log P(\mathbf{x}, \mathbf{y}|\theta)$$

We are not just interested in estimating only the parameters but also the structure of MLNs. Note that our approach for learning the structure is non-parametric, i.e. we do not have a fixed set of parameters θ but instead use a regression function ψ. This is a subtle yet important distinction to the EM learning methods for SRL (Natarajan et al, 2008; Jaeger, 2007; Xiang and Neville, 2008) that estimate the parameters given a fixed structure. The relational regression trees of our models define the structure of the potential function and the leaves represent the parameters of these potentials. We sum the likelihood function over all possible hidden groundings to compute the marginal probabilities of the observed ones.

$$\ell(\psi) \equiv \log \sum_{\mathbf{y} \in \mathcal{Y}} P(\mathbf{x}; \mathbf{y}|\psi) = \log \sum_{\mathbf{y} \in \mathcal{Y}} \left\{ P(\mathbf{y}|\mathbf{x}; \psi_t) \frac{P(\mathbf{x}; \mathbf{y}|\psi)}{P(\mathbf{y}|\mathbf{x}; \psi_t)} \right\}$$

Similar to the fully observed case, ψ is the regression function that is used to calculate $P(X = x|\mathbf{MB}(x))$ (see Equation 4). After t iterations of the EM steps, ψ_t represents the current model and we must update this model by finding a ψ that maximizes the log-likelihood. Unfortunately maximizing the $\ell(\psi)$ function directly is not feasible and hence we find a lower bound on $\ell(\psi)$. Applying Jensen's inequality on the loglikelihood defined above,

$$\begin{aligned}\ell(\psi) &\geq \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t)\log\frac{P(\mathbf{x};\mathbf{y}|\psi)}{P(\mathbf{y}|\mathbf{x};\psi_t)} = \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t)\log\frac{P(\mathbf{x};\mathbf{y}|\psi)P(\mathbf{x}|\psi_t)}{P(\mathbf{x};\mathbf{y}|\psi_t)} \\ &= \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t)\log P(\mathbf{x};\mathbf{y}|\psi) - \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t)\log P(\mathbf{x};\mathbf{y}|\psi_t) \\ &+ \log P(\mathbf{x};\psi_t)\sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t) \\ &= \mathcal{Q}(\psi;\psi_t) - \mathcal{Q}(\psi_t;\psi_t) + \ell(\psi_t)\end{aligned}$$

where $\mathcal{Q}(\psi;\psi_t) \equiv \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t)\log P(\mathbf{x};\mathbf{y}|\psi)$. Instead of finding ψ that would maximize $\ell(\psi)$, it would be easier to find the ψ that would maximize this lower bound. Since ψ_t is constant with respect to the parameter ψ, we only need to find the ψ that would maximize $\mathcal{Q}(\psi;\psi_t)$. However, in many situations, finding a ψ that improves over $\mathcal{Q}(\psi;\psi_t)$ could suffice as shown next in the theorem.

Theorem

Any approach that finds ψ which improves over $\mathcal{Q}(\psi;\psi_t)$ guarantees a monotonic increase in the log-likelihood of the observed data.

Proof Sketch

Consider ψ_{t+1} obtained in the $(t+1)$th iteration that improves over ψ_t, i.e.

$$\begin{aligned}\mathcal{Q}(\psi_{t+1};\psi_t) \geq \mathcal{Q}(\psi_t;\psi_t) \Rightarrow &\mathcal{Q}(\psi_{t+1};\psi_t) - \mathcal{Q}(\psi_t;\psi_t) &\geq 0 \\ \Rightarrow &\mathcal{Q}(\psi_{t+1};\psi_t) - \mathcal{Q}(\psi_t;\psi_t) + \ell(\psi_t) &\geq \ell(\psi_t)\end{aligned}$$

Since $\ell(\psi_{t+1})$ is lower bounded by $\mathcal{Q}(\psi_{t+1};\psi_t) - \mathcal{Q}(\psi_t;\psi_t) + \ell(\psi_t)$,

$$\ell(\psi_{t+1}) \geq \mathcal{Q}(\psi_{t+1};\psi_t) - \mathcal{Q}(\psi_t;\psi_t) + \ell(\psi_t)$$

Combining these two equations, we get

$$\ell(\psi_{t+1}) \geq \mathcal{Q}(\psi_{t+1};\psi_t) - \mathcal{Q}(\psi_t;\psi_t) + \ell(\psi_t) \geq \ell(\psi_t)$$

Hence the log-likelihood value increases or stays the same after every M step. ■

Recall that as we use the pseudo-loglikelihood in $\mathcal{Q}(\psi;\psi_t)$, we rewrite $P(\mathbf{x};\mathbf{y}|\psi)$ as $\prod_{z\in\mathbf{x};\mathbf{y}} P(z|\mathbf{z}_{-z};\psi)$. We define $\mathbf{Z}$ as the union of all the predicates, i.e. $\mathbf{Z} = \mathbf{X}\cup\mathbf{Y}$. We use $\mathbf{z}_{-z}$ to denote $\mathbf{z}\setminus z$ and $\mathcal{Y}_{-i}$ to represent the world states for the set of groundings $\mathbf{y}_{-y_i}$ (i.e. $\mathbf{y}\setminus y_i$). Hence we can rewrite $\mathcal{Q}(\psi;\psi_t)$

$$\mathcal{Q}(\psi;\psi_t) = \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t)\sum_{z\in\mathbf{x}\cup\mathbf{y}}\log P(z|\mathbf{z}_{-z};\psi)$$

Since we do not have a closed form solution for ψ, we use functional gradient descent. Following generalized EM algorithms (Dempster et al, 1977) rather than finding the maximum, we take S gradient steps in the M-step. This allows us to amortize the cost of sampling the states and run enough iterations in reasonable

Algorithm 4 updateModel(W, ψ)

```
S := 2                                          ▷ Number of trees learned in M-step
for i ≤ S do                                    ▷ Iterate over target and hidden predicates, P
    for p ∈ P do                                ▷ E_p := Downsampled groundings of p
        D_p := buildDataset(E_p, W, ψ)
        T_p := learnTree(D_p)
        ψ = ψ + T_p
    end for
end for
return ψ
```

time without making the model too large. While learning MLNs, each gradient step was approximated by a single tree. Thus, we learn S trees in every M-step.

We present our proposed approach for updating the model in Algorithm 4. We iterate through all the query and hidden predicates and learn one tree for each predicate. We compute the gradients for the groundings of predicate p given by E_p, using the world states W and current model ψ. We then learn a relational regression tree using this dataset and add it to our current model. The $learnTree$ function uses different scoring functions depending on the model as we show later. The set E_p may not contain all the groundings of the predicate p, since we downsample the negative examples during every iteration by randomly selecting the negatives so that there are twice as many negative as positive examples. Relational datasets generally have many more negatives than positives and learning methods perform better if the majority class is downsampled (Chan and Stolfo, 1998).

Next, we need to compute the gradients for each example (i.e. hidden and observed groundings of the target and hidden predicates) which will be used to learn the next regression tree (T_p). The value returned by the ψ function also depends on other ground literals, since their values will influence the path taken in the regression tree. In the previous section, we included them as arguments to the function definition, i.e. $\psi(x; \mathbf{MB}(x))$. But $\mathbf{MB}(x)$ is observed and has the same values across all examples (the blanket varies across examples but the ground literal values are the same) and so the function can be simplified to $\psi(x)$. However, with missing data, the assignment to the hidden variables $\mathbf{y}$ is not constant as each assignment to $\mathbf{y}$ may return a different value for a given example (due to different paths). Hence, we include the assignment to the hidden variables in our function ($\psi(x; \mathbf{y})$) and compute the gradients for an example and hidden state assignment.

4.2 Gradients for hidden groundings

We now derive the gradients of $\mathcal{Q}$ w.r.t the hidden groundings by taking partial derivatives of $\mathcal{Q}$ w.r.t $\psi(y_i; \mathbf{y}_{-i})$ where y_i is a hidden grounding. The value of $\psi(y_i; \mathbf{y}_{-i})$ is only used to calculate $P(y_i|\mathbf{x}, \mathbf{y}_{-i}; \psi)$ for two world states: where y_i is true and where y_i is false. So the gradient w.r.t. $\psi(y_i; \mathbf{y}_{-i})$ can be calculated as

$$\begin{aligned}\frac{\partial \mathcal{Q}(\psi; \psi_t)}{\psi(y_i; \mathbf{y}_{-i})} &= P(y_i = 1, \mathbf{y}_{-i}|\mathbf{x}; \psi_t) \frac{\partial \log P(y_i = 1|\mathbf{x}, \mathbf{y}_{-i}; \psi)}{\partial \psi(y_i; \mathbf{y}_{-i})} \\ &+ P(y_i = 0, \mathbf{y}_{-i}|\mathbf{x}; \psi_t) \frac{\partial \log P(y_i = 0|\mathbf{x}, \mathbf{y}_{-i}; \psi)}{\partial \psi(y_i; \mathbf{y}_{-i})}\end{aligned}$$

As shown before, the gradients correspond to the difference between the true value of y_i and the current predicted probability of y_i (i.e. $I(y_i = y) - P(y_i = y)$). As we have terms involving $P(y_i)$ for each value of y_i, we get two gradient terms.

$$\begin{aligned} & P(y_i = 1, \mathbf{y}_{-i}|\mathbf{x}; \psi_t)(1 - P(y_i = 1|\mathbf{x}, \mathbf{y}_{-i}; \psi)) \\ + \quad & P(y_i = 0, \mathbf{y}_{-i}|\mathbf{x}; \psi_t)(0 - P(y_i = 1|\mathbf{x}, \mathbf{y}_{-i}; \psi)) \\ = \quad & P(y_i = 1, \mathbf{y}_{-i}|\mathbf{x}; \psi_t) - P(\mathbf{y}_{-i}|\mathbf{x}; \psi_t)P(y_i = 1|\mathbf{x}, \mathbf{y}_{-i}; \psi)) \end{aligned} \tag{7}$$

With the PLL assumption, the gradients can be written as $\prod_{j \neq i} P(y_j|\mathbf{x}, \mathbf{y}_{-j}; \psi_t)$ $[P(y_i = 1|\mathbf{x}, \mathbf{y}_{-i}; \psi_t) - P(y_i = 1|\mathbf{x}, \mathbf{y}_{-i}; \psi)]$. Intuitively, the gradients correspond to the difference between the probability predictions weighted by the probability of the hidden state assignment.

4.3 Gradients for observed groundings

To compute the gradients for the observed groundings, we take partial derivatives of $\mathcal{Q}$ with respect to $\psi(x_i; \mathbf{y})$ where x_i is observed in the data. Similar to the gradients for hidden groundings, we use $\mathbf{y}$ as an argument in the ψ function and only consider the world states that matches with the given argument. The gradient w.r.t. $\psi(x_i; \mathbf{y})$ can be calculated as

$$\frac{\partial \mathcal{Q}(\psi; \psi_t)}{\psi(x_i; \mathbf{y})} = P(\mathbf{y}|\mathbf{x}; \psi_t) \frac{\partial \log P(x_i|\mathbf{x}_{-i}, \mathbf{y}; \psi)}{\partial \psi(x_i; \mathbf{y})} \tag{8}$$

$$= P(\mathbf{y}|\mathbf{x}; \psi_t)[I(x_i) - P(x_i = 1|\mathbf{z}_{-x_i}; \psi)] \tag{9}$$

Similar to the hidden groundings, the gradients correspond to the difference between the predictions weighted by the probability of the hidden state assignment.

4.4 Regression tree learning

The input examples to our regression tree learner are of the form $< (z; \mathbf{y}), \Delta >$. For every ground literal $z \in \mathbf{x} \cup \mathbf{y}$, we calculate the gradients for an assignment to the hidden variables. Algorithm 5 describes the *buildDataset* function used to generate these examples. For every ground literal e and every world state w (i.e., $\mathbf{y}$), we compute the gradient of the example ($gradient(e, w)$). For examples that are observed, we use equation 9 to compute $gradient(e, w)$ and for examples that are hidden, we use equation 7. Similar to our structure learning approach, we use only

Algorithm 5 buildDataset(E_p, W, ψ)

```
1: D_p := ∅
2: for e ∈ E_p do
3:     for w ∈ W do
4:         Δ_e := gradient(e, w)
5:         D_p := D_p ∪ < (e; w), Δ_e >
6:     end for
7: end for
8: return D_p
```

a subset of the examples for learning the regression function. Apart from subsampling the ground literals, we also pick $|W|$ hidden state assignments from $\mathcal{Y}$. Since our gradients are weighted by the probability of the hidden state assignment $\mathbf{y}$, an unlikely assignment will result in small gradients and thereby have little influence on the learned tree. Hence, we use Gibbs sampling to sample the most likely hidden state assignments. Also we approximate the joint probability of an hidden state assignment with the pseudo-likelihood, i.e. $P(\mathbf{y}|\mathbf{x};\psi_t) = \prod_i P(y_i|\mathbf{x},\mathbf{y}_{-i};\psi_t)$.

To summarize, we learn RRTs for the gradients presented with the modified scoring function. To compute the marginal probability of any example, trees for all the predicates would be used. Hence while learning, a single tree is learned for each predicate and the gradients are computed based on the trees learned till the current iteration. We resample the hidden states after two such iterations over the target and hidden predicates.

4.5 Adapting RFGB-EM for other SRL models

Natarajan et al. (2012) describe learning RDNs using functional-gradient boosting where all the trees are learned for a target predicate before the next predicate. Since the gradients for each predicate are independent of the model for other predicates, one can learn all the trees independently. We, on the other hand, update the hidden world states after every two iterations (note $S = 2$) and hence for every predicate we learn two trees at a time. We then resample the hidden states and use the sampled states for the next two iterations. Unlike RFGB for MLNs presented before, we use RRTs with the weighted variance scoring function for fitting the gradients for each example. The $learnTree$ function in Algorithm 4 can use any off-the-shelf learner. The key difference to the approach by Natarajan et al. (2012) is that we learn only two trees at a time and iterate through all the predicates.

For imitation learning, similar to the RDN case, we are learning the distribution over the actions for every state using the training trajectories provided. The set of predicates, P contains all the action predicates and the hidden predicates. We can then learn RRTs to predict each action while updating the hidden values. Natarajan et al. (2011) learned all the trees for each action independently whereas we learn two trees for every predicate before resampling the hidden ones.

5 Experimental Results

We now present the results of experiments in two settings (1) RFGB for MLNs with no hidden data and (2) RFGB-EM for MLNs, RDN and imitation learning in the presence of hidden data[2]. For measuring performance across multiple approaches and multiple data sets, generally we use the AUC-PR (Area under the Precision-Recall curve) and CLL (Conditional log-likelihood) values. A major strength of PR curves is that they ignore the impact of correctly labeling negative examples and instead focus on the typically rarer and yet more important, positive examples. Hence, we not only present the CLL values, but also the AUC-PR values that are shown to be more conservative estimate of the performance compared to AUC-ROC (Davis and Goadrich, 2006).

[2] Software and datasets available at http://pages.cs.wisc.edu/∼tushar/Boostr/

In all our experiments, we present the numbers in bold whenever they are statistically significantly better than the baseline approaches. We used the paired t-test with p-value=0.05 for determining statistical significance. Additional details specific to individual experiments are described in the respective subsections.

5.1 Experimental Results - Fully Observable Case

We compare our two boosting algorithms - tree-based (MLN-BT) and clause-based (MLN-BC) to four state-of-the-art MLN structure learning methods: LHL (Kok and Domingos, 2009), BUSL (Mihalkova and Mooney, 2007), Motif-S (short rules) and Motif-L (long rules) (Kok and Domingos, 2010) on three datasets. Additional experiments on Cora dataset can be found in our previous paper (Khot et al, 2011). We employed the default settings of Alchemy (Kok et al, 2010) for weight learning on all the datasets, unless mentioned otherwise. We set the *multipleDatabases* flag to true for weight learning. For inference, we used MC-SAT sampling with 1 million sampling steps or 24 hours whichever occurs earlier. For learning structure using motifs, we used the settings provided by Kok and Domingos (Kok and Domingos, 2010). While employing LHL and BUSL for structure learning, we used the default settings in Alchemy. We set the maximum number of leaves in MLN-BT to 8 and maximum number of clauses to 3 in MLN-BC. The beam-width was set to 10 and maximum clause length was set to 3 for MLN-BC. We used 20 gradient steps on all the boosting approaches. Since we ran the experiments on a heterogenous cluster of machines, we do not report the learning time for the larger IMDB dataset. For the Alchemy-based structure-learning algorithms, we tried several different weight learning methods such as conjugate gradient and voted perceptron. We then present the ones with the best results.

5.1.1 UW-CSE dataset

The goal in the UW data set (Richardson and Domingos, 2006) is to predict the `advisedBy` relationship between a student and a professor. The data set consists of details of professors, students and courses from five different sub-areas of computer science (AI, programming languages, theory, system and graphics). Predicates include `professor, student, publication, advisedBy, hasPosition, projectMember, yearsInProgram, courseLevel, taughtBy` and `teachingAssistant`. Our task is to learn using the other predicates, to predict the `advisedBy` relation. We employ five-fold cross validation where we learn from four areas and predict on the other area. We also compared against the handcoded MLN available on Alchemy's website with discriminative weight learning (shown as *Alchemy-D* in the tables). We were unable to get BUSL to run due to segmentation fault issues.

Table 1 presents the AUC and CLL values, along with the training time taken by each method averaged over five-folds. The training time does not change for the different test-sets. As can be seen, for the complete dataset both boosting approaches (MLN-BT and MLN-BC) perform better than other MLN learning techniques on the AUC-PR values. Current MLN learning algorithms on the other hand are able to achieve lower CLL values over the complete dataset by pushing the probabilities to 0, but are not able to differentiate between positive and negative examples as shown by the low AUC-PR values.

Table 1: Results on UW data set

Algo	2X negatives		All negatives		Training
	AUC-PR	CLL	AUC-PR	CLL	Time
MLN-BT	$\mathbf{0.94 \pm 0.06}$	$\mathbf{-0.52 \pm 0.45}$	0.21 ± 0.17	-0.46 ± 0.36	18.4 sec
MLN-BC	$\mathbf{0.95 \pm 0.05}$	$\mathbf{-0.30 \pm 0.06}$	0.22 ± 0.17	-0.47 ± 0.14	33.3 sec
Motif-S	0.43 ± 0.03	-3.23 ± 0.78	0.01 ± 0.00	-0.06 ± 0.03	1.8 hrs
Motif-L	0.27 ± 0.06	-3.60 ± 0.56	0.01 ± 0.00	-0.07 ± 0.02	10.1 hrs
Alchemy-D	0.31 ± 0.10	-3.90 ± 0.41	0.01 ± 0.00	-0.08 ± 0.02	7.1 hrs
LHL	0.42 ± 0.10	-2.94 ± 0.31	0.01 ± 0.01	-0.06 ± 0.02	37.2 sec

When we reduce the negatives in the test set to twice the number of positives, the boosting techniques dominate on both the AUC-PR and CLL values, while the other techniques, which cannot differentiate between the examples, have poor CLL values. Also, there is no significant difference between learning the trees or the clauses in the case of boosting MLNs. On this domain, RDNs are able to model the data better than MLNs. Boosted RDNs achieve an AUC-PR of 0.95 ± 0.03 and CLL of -0.17 ± 0.03 for 2X negatives (Natarajan et al, 2012).

We performed additional experiments on this data set to understand the impact of number of trees on the predictive performance. Figures 3 and 4 present the CLL and AUC-PR values averaged over 30 runs as a function of the number of trees. As can be seen, CLL values improve as the number of trees increase. This is due to the fact that adding more trees amounts to moving the likelihood of the examples towards 1. On the other hand, the AUC-PR values increase for the first few trees. After a small amount of trees (in this case around 6), the value has plateaued. In all our experiments, we observed that increasing the number of trees beyond 20 had no significant impact in AUC-PR values. Our results show that with a small number of trees, the boosting-based methods are able to achieve reasonable predictive performance.

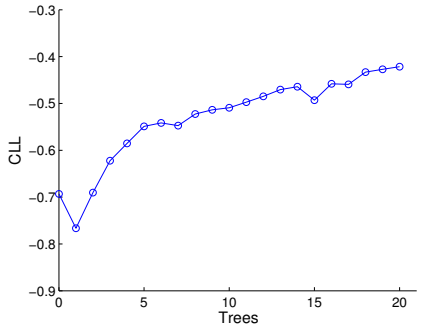


Fig. 3: Num trees vs. CLL for UW

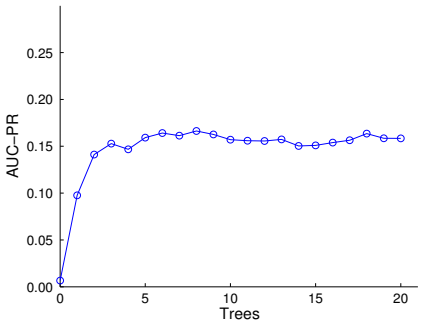


Fig. 4: Num trees vs. AUC-PR for UW

5.1.2 IMDB dataset

The IMDB dataset was first used by Mihalkova and Mooney (2007) and contains five predicates: `actor, director, genre, gender` and `workedUnder`. We do not evaluate the `actor` and `director` predicates as they are mutually exclusive facts in this dataset and easy to learn for all the methods. Also since gender can take

only two values, we convert the `gender(person,gender)` predicate to a single argument predicate `female_gender(person)`. Following Kok and Domingos (2009), we omitted the four equality predicates. Our goal was to predict `workedUnder, genre` and `gender` given all the other predicates as evidence. We conducted five-fold cross-validation and averaged the results across all the folds. We perform inference over every predicate given all other predicates as evidence.

Table 2: Results on IMDB data set

	AUC-PR			CLL		
Algorithm	workedUnder	genre	gender	workedUnder	genre	gender
MLN-BT	0.90 ± 0.07	0.94 ± 0.08	0.45 ± 0.06	-0.18 ± 0.06	-0.20 ± 0.09	-0.62 ± 0.05
MLN-BC	1.00 ± 0.00	1.00 ± 0.00	0.39 ± 0.07	-0.11 ± 0.04	-0.12 ± 0.08	-0.84 ± 0.21
RDN-B	0.99 ± 0.02	0.91 ± 0.12	0.46 ± 0.18	-0.88 ± 0.20	-0.25 ± 0.22	-0.76 ± 0.16
BUSL	0.89 ± 0.11	0.94 ± 0.08	0.44 ± 0.08	-0.56 ± 0.05	-0.27 ± 0.09	-0.69 ± 0.01
LHL	1.00 ± 0.00	0.37 ± 0.09	0.39 ± 0.12	-0.02 ± 0.01	-1.13 ± 0.23	-0.73 ± 0.05
Motif-S	0.56 ± 0.16	0.52 ± 0.29	0.48 ± 0.08	-2.73 ± 1.66	-3.99 ± 2.70	-0.71 ± 0.08
Motif-L	0.48 ± 0.27	0.39 ± 0.03	0.46 ± 0.08	-2.30 ± 1.16	-2.32 ± 1.15	-0.69 ± 0.06

Table 2 shows the AUC values for the three predicates: `workedUnder, genre` and `gender`. The boosting approaches perform better on average, on both the AUC and CLL values, than the other methods. The BUSL method seems to exhibit the best performance of the prior structure-learning methods in this domain. Our boosting algorithms seem to be comparable or better than BUSL on all the predicates. For `workedUnder`, LHL has comparable AUC values to the boosting approaches, while it is clearly worse on the other predicates. There is no significant difference between the two versions of the boosting algorithms.

The other question that we consider in this domain is: how do boosted MLNs compare against boosted RDNs (Natarajan et al, 2012)? To answer this question, we compared our proposed methods against boosted RDNs (RDN-B). As can be seen from Table 2, the MLN-based methods are marginally better than the boosted RDNs for predicting `workedUnder` predicate, while comparable for others.

5.1.3 WebKB dataset

The WebKB dataset was first created by Craven et al. (1998) and contains information about department webpages and the links between them. It also contains the categories for each webpage and the words within each page. This dataset was converted by Mihalkova and Mooney(2007) to contain only the category of each webpage and links between these pages. They created the following predicates: `Student(A), Faculty(A), CourseTA(C, A)`, `CourseProf(C, A), Project(P, A)` and `SamePerson(A, B)` from these webpages. The textual information was ignored. We removed the `SamePerson(A, B)` predicate as it only had groundings with both the arguments being exactly same (i.e., `SamePerson(A,A)`).

We evaluated all the methods over the `CourseProf` and `CourseTA` predicates since all other predicates had trivial rules such as `courseTA(C,A)` $\rightarrow$ `Student(A)`. We performed four-fold cross-validation where each fold corresponds to one university. We do not present the performance of BUSL and Motif-S because the algorithms were unable to learn any useful rules and had a AUC-PR value of 0.

Table 3: Results on the WebKB data set with all negatives

	AUC-PR		CLL	
	courseTA	courseProf	courseTA	courseProf
MLN-BT	0.005 ± 0.003	0.029 ± 0.005	$\mathbf{-0.359 \pm 0.041}$	$\mathbf{-0.334 \pm 0.068}$
MLN-BC	0.004 ± 0.002	0.027 ± 0.007	$\mathbf{-0.479 \pm 0.041}$	$\mathbf{-0.304 \pm 0.010}$
LHL	0.004 ± 0.002	0.007 ± 0.002	-0.023 ± 0.011	-0.029 ± 0.005
Motif-L	0.003 ± 0.002	0.017 ± 0.009	-0.024 ± 0.011	-0.029 ± 0.005

Table 4: Results on the WebKB data set with 2x negatives

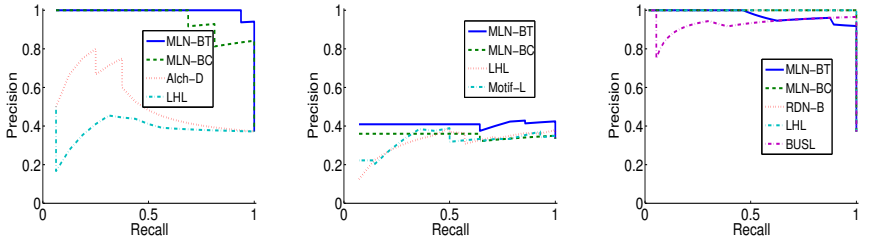
	AUC-PR		CLL	
	courseTA	courseProf	courseTA	courseProf
MLN-BT	0.426 ± 0.027	0.738 ± 0.034	$\mathbf{-0.603 \pm 0.057}$	$\mathbf{-0.406 \pm 0.050}$
MLN-BC	0.379 ± 0.031	0.750 ± 0.110	$\mathbf{-0.656 \pm 0.012}$	$\mathbf{-0.357 \pm 0.045}$
LHL	0.350 ± 0.046	0.460 ± 0.036	-2.274 ± 0.102	-2.243 ± 0.104
Motif-L	0.332 ± 0.014	0.637 ± 0.219	-2.282 ± 0.110	-2.198 ± 0.105

Table 3 and 4 presents the results of the different algorithms in this domain. As with UW data set we present two different cases here. Table 3 uses the data set with all the negative examples in the test set and Table 4 uses the data set with twice the number of negatives as positives. Similar to the earlier case, in the test set with all negatives, current MLN methods such as LHL and Motifs exhibit good performance for the CLL evaluation measure for both the `courseTA` and `courseProf` predicates. On the other hand, the AUC-PR values are lower than that of our boosting-based methods. This difference is magnified when we limit the number of negatives to twice the number of positives. In the latter case, even the CLL for the current MLN structure learning algorithms are significantly worse than our boosting methods. There is no statistically significant difference between the performance of the boosting methods. Our current results show that employing a test set with a reasonable distribution of the classes yields a better insight into the difference in the performance of the learning algorithms. In terms of average learning time for these approaches, MLN-BC takes 22 seconds, MLN-BT takes 47.5 seconds and LHL (fastest baseline in UW-CSE) takes 60.5 seconds.

Precision-Recall curves We also present the PR curves for the first fold on the `advisedBy` predicate in UW in Figure 5(a), the `courseTA` predicate in Web-KB in Figure 5(b), and `workedUnder` predicate in IMDB in Figure 5(c). We only show the curves for the best previously published structure-learning methods. Our algorithms exhibit a clear superior performance especially in the high-recall regions.

5.2 Experimental Results - Hidden Data Case

We now present the results of our EM approach on four different problems for the partially observable data case. We present results for structural EM (SEM) with a suffix to indicate the number of hidden state samples used, i.e. $|W|$ mentioned in



(a) UW - advisedBy (b) WebKB - CourseTA (c) IMDB - WorkedUnder

Fig. 5: PR Curves for Some Domain - Predicate Pairs

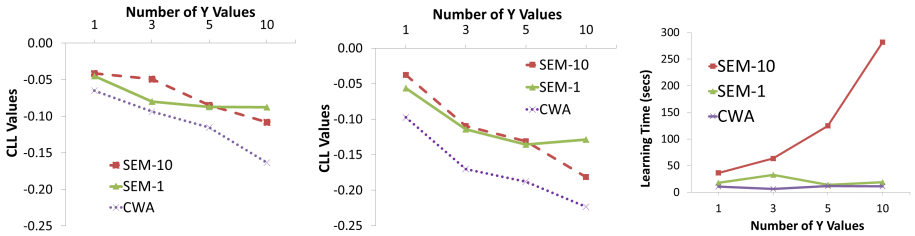
the Section 4 (e.g. *SEM-10* uses ten samples while *SEM-1* uses the single MAP estimate). *SEM-10* corresponds to the soft-EM approach whereas *SEM-1* corresponds to the hard-EM approach. We also present the results of using RFGB without using EM while setting all hidden groundings to false, i.e. using the closed world assumption (denoted as *CWA*). These methods are essentially the prior work on RDNs (Natarajan et al, 2012), MLNs (Section 3) and imitation learning (Natarajan et al, 2011). Each of these methods were run for 10 gradient iterations. In these experiments we attempt to empirically investigate the following questions:
Q1: Can removing CWA for relational structure learning improve the performance?
Q2: Can soft-EM outperform hard-EM in relational domains?

5.2.1 Disjunctive dataset

We generated a simple synthetic dataset to compare *SEM* against *CWA* using RDNs as the base model. We used three predicates *q(X, Y)*, *r(X, Y)* and *s(X)*. The range of X was $1, ..., 100$, and varied Y to have four different values $|Y| \in \{1, 3, 5, 10\}$ as shown in Figure 6. We treated the predicate r as hidden and the goal was to predict s. To generate the training data, we used a distribution $P(r|q)$. We then combine `r(X,Y)` for different values of Y using an OR condition to generate `s(X)`. Hence `s(X)` given `r(X,Y)` is a deterministic rule where `s(X)` is true if for some Y, `r(X,Y)` is true. We generated 10 synthetic datasets with randomly sampled hidden data, trained one model on each dataset and evaluated each model on the other nine datasets. We average the results from all these runs. We used this synthetic dataset as it allows us to evaluate approaches against varying importance of accurately predicting the missing data.

The results on this domain are presented in Figure 6 (higher is better). We hide 20% and 40% of the groundings of the hidden predicates to simulate different levels of missing data. We only present the CLL values since the AUC-PR values are nearly equal for all the approaches. The EM approaches outperform CWA in all scenarios thereby affirmatively answering *Q1* for this domain. *SEM-10* outperforms both *SEM-1* and *CWA* methods on this dataset for $|Y| = 1$ and $|Y| = 3$, whereas *SEM-1* outperforms the others for $|Y| = 10$. Although the difference is very small in some cases, it is statistically significant (except for $|Y| = 5$ where *SEM-10* has similar performance to *SEM-1*).

As we increase the number of values that can be taken by Y, we increase the number of possible hidden states. With just 10 samples, *SEM-10* is able to capture a relatively large space of the possible assignments to the hidden states for $Y = 1$



(a) 20% Missing data (b) 40% Missing data (c) Avg.learning time

Fig. 6: Results on the Disjunctive dataset.

and $Y = 3$. On the other hand for $Y = 10$, both *SEM-10* and *SEM-1* capture a very small number of hidden state assignments compared to the total number of possible assignments. As a result, the simpler *SEM-1* is able to perform better when $|Y| = 10$. Increasing the number of sampled states for soft-EM would improve the performance but at the cost of learning time. We also present the change in learning time of the various approaches with increasing missing data (by varying the $|Y|$ values) in Figure 6(c). We averaged the learning times across the ten folds for 20% missing data. Both *SEM-1* and *CWA* have similar learning times showing that the sampling step for this dataset is not computationally intensive. Since we used a heterogeneous cluster of machines, a slower machine may introduce a small bump, viz. at $|Y| = 3$. On the other hand, learning time for *SEM-10* increases exponentially with increase in the number of missing values. This is primarily due to the fact that every node in the tree has to be scored for every sampled world state in W (by adding and removing the required facts every time). As the number of hidden groundings increase, the size of each sampled state increases requiring more operations during the learning phase. Increasing $|W|$ would further increase the learning time but could improve the accuracy.

5.2.2 Cancer dataset: MLN Structure Learning

The cancer MLN is a popular synthetic data set (Kersting et al, 2009; Domingos and Lowd, 2009). We created a friend network using a symmetric predicate, `friends(X,Y)`. Each person has three attributes: `stress(X)`, `cancer(X)` and `smokes(X)`. The stress attribute for each person is set using a Bernoulli distribution. A person is more likely to smoke if he is stressed (set using a Bernoulli distribution) or has friends who smoke (set using an exponential distribution). Similarly, a person is likely to have cancer if he smokes (set using a Bernoulli distribution) or he has a lot of friends who smoke (set using an exponential distribution). The more smoker friends a person has, the more likely he is to get cancer. Such rules can be captured by MLNs since the prob-

Table 5: Results on Cancer dataset

Hidden %	20%		40%	
Algorithm	CLL	AUC-PR	CLL	AUC-PR
SEM-10	**-1.445**	**0.482**	**-1.315**	**0.510**
SEM-1	-1.648	**0.483**	-1.586	0.500
CWA	-1.629	0.478	-1.693	0.488

abilities are proportional to the number of groundings of a clause (e.g. $smokes(y) \wedge friend(x, y) \rightarrow smokes(x)$). The target predicate is `cancer` while `smokes` has some missing groundings. We trained the model on 10 generated datasets with randomly sampled hidden data and evaluated each model on other nine datasets and present the average results.

As seen in Table 5, *SEM-10* mostly outperforms the other approaches both in terms of CLL and AUC-PR. For 20% missing data, there is no statistically significant difference between the two EM approaches but both methods outperform CWA. Unlike the previous domains, SEM-10 is at least as good as or better than SEM-1 in this domain. Hence for this domain, we can affirmatively answer both *Q1* and *Q2*. Since Alchemy does not have a mechanism to handle missing data for structure learning, we ran weight learning (generative with 10000 iterations and 1e-5 threshold) on hand-written rules and learned the weights using the missing data weight learning approach of Alchemy. The AUC PR values were around 0.6. This shows that simply learning the parameters is reasonably comparable to our models that learn both the structure and parameters with hidden data.

5.2.3 UW-CSE dataset: RDN Structure Learning

We use the UW-CSE dataset described before in Subsection 5.1.1 to evaluate the EM approach for learning RDNs. We randomly hid groundings of the `tempAdvisedby`, `inPhase` and `hasPosition` predicates during training. Due to these hidden groundings and the different type of SRL model being learned, our numbers are not exactly comparable to the ones reported in Section 5.1. We performed five-fold cross-validation and present the CLL values in Table 6. We do not present the AUC PR values since the difference is not statistically significant. We also varied the amount of hidden data in our experiments ("Hidden %" in the table indicates the percentage of the groundings being hidden).

Table 6: CLL values for UW-CSE

Hidden %	20%	40%
SEM-10	**-0.168**	**-0.170**
SEM-1	**-0.150**	**-0.151**
CWA	-0.187	-0.192

Table 7: CLL values for IMDB

Hidden %	10%	20%
SEM-10	**-0.501**	**-0.551**
SEM-1	**-0.423**	**-0.467**
CWA	-0.586	-0.80

In general, the EM methods perform statistically significantly (with p-value < 0.05) better than the closed world assumption. Hence, we can answer *Q1* affirmatively in this real world domain too. It appears that in this domain, using a single sample for the hidden state has the same performance as that of using 10 samples. This is in line with most EM algorithms where using a single state (MAP) approximation generally suffices (negatively answering *Q2* in this domain).

5.2.4 IMDB dataset: RDN Structure Learning

We also use the IMDB dataset described before (Subsection 5.1.2) to evaluate the EM approach. We predicted the `gender` predicate given all the other predicates.

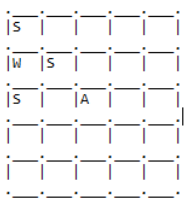


Fig. 7: Wumpus world. W is the wumpus, S is the stench and A is the agent.

Table 8: Results for Wumpus dataset

Hidden %	20%		40%	
Algorithm	CLL	AUC-PR	CLL	AUC-PR
SEM-10	**-0.245**	**0.857**	**-0.261**	**0.853**
SEM-1	-0.278	0.845	-0.283	0.839
CWA	-0.282	0.826	-0.270	0.826

We randomly hid the groundings of `actor` and `workedUnder` predicates during learning and inference. Again due to these hidden predicates, our numbers are not comparable to the ones reported earlier. We performed five-fold cross-validation.

We present the CLL values for hiding 10% and 20% of the groundings of the two hidden predicates in Table 7. Similar to the disjunctive dataset, there is no statistically significant difference between the three methods in the AUC-PR values and hence are not reported here. In general, the EM methods perform statistically significantly (with p-value < 0.05) better than the closed world assumption. Hence we can again affirmatively answer *Q1* in this domain. Between the two EM methods, using one sample is sufficient to capture the underlying distribution and hence the simpler SEM-1 has a higher CLL value than SEM-10.

5.2.5 Wumpus world: Relational Imitation Learning

We also performed imitation learning in a partially observed relational domain where we created a simple version of the Wumpus task. The location of wumpus is partially observed. We used a 5×5 grid with the wumpus placed at a random location. The wumpus is always surrounded by stench on all four sides.

Figure 7 shows one instantiation of the initial grid locations. The agent can perform 8 possible actions: 4 move actions in each direction and 4 shoot actions in each direction. The agent's task is to move to a cell such that he can fire an arrow to kill the wumpus. The dataset contains predicates for each cell such as `cellAt, cellRight` and `cellAbove` and obstacle locations such as `wumpus` and `stench`. The wumpus is not observed in all the trajectories although the stench is always observed. Trajectories were created by human users whose policy generally is to move towards the wumpus' row or column and shoot accordingly.

The EM approaches (using the trajectories where wumpus is observed) learn that wumpus is surrounded by stench and fill the missing values in other trajectories. The *CWA* approach (Natarajan et al, 2011) on the other hand assumes that the wumpus is not present and relies on the stench to guess the action to be performed. The results are presented in Table 8. From the results, it can be easily observed that the EM methods are superior to that of the prior work on imitation learning. Moreover, *SEM-10* which uses multiple samples outperforms the single-sample *SEM-1* approach. This domain clearly shows that the previous method of boosting in imitation learning is not sufficient in problems with partial observability and it is imperative to employ methods that do not assume closed-world. Similar to the Cancer domain, we can affirmatively answer *Q1* and *Q2*.

In conclusion, our experiments have shown that opening the closed-world assumption definitely results in an improvement in the performance. Between the two EM approaches, we have shown empirically that for certain domains (e.g. UW, IMDB) a single sample (hard-EM) might be sufficient, whereas in certain domains (e.g. Cancer, Wumpus) multiple samples (soft-EM) are needed to capture the true distribution. Thus, both the questions can generally be answered affirmatively (where answer to *Q2* depends on the domain).

But these improvements come at the cost of increased learning time where SEM-10 can be comparatively much slower than SEM-1. SEM-10 can take from 15 hrs (Wumpus) to 22 minutes (UW-CSE) where SEM-1 takes 3 minutes on both of these datasets. Comparatively the CWA approaches take 1 minute to learn the model on all of these datasets. Since the gradients are computed for every example and hidden state in SEM-10, the number of examples grow to $n \times 10$ where n is the number of examples in CWA. Moreover we need to manipulates the facts based on every hidden state for every candidate node during tree learning.

6 Discussion and Future Work

Due to the ability to write rules easily whose weights can be learned with efficent algorithms and the presence of convergent inference approaches (Singla and Domingos, 2008), MLNs are popular. But learning the structure of MLNs remains one of the hardest and challenging problems. We address this problem by using gradient-boosting with the added benefit of learning weights simultaneously. Building upon the success of pseudo-likelihood methods for MLNs, we derived tree-based and clause-based gradient boosting algorithms. We evaluated the algorithms on four standard datasets and established the superior performance of the boosting method across all the domains and all the predicates.

One future direction for the structure learning approach is to derive the functional gradients for the full likelihood instead of the pseudo-likelihood and learn the trees/clauses for jointly predicting several predicates. Another direction is to induce a simpler MLN that approximates the learned set of clauses/trees; this will ensure that the learned model is interpretable as well. Another important direction is to evaluate the scaling properties of the algorithm in large data sets.

We also addressed the challenging problem of learning SRL models in the presence of hidden data. We developed an EM-based algorithm for functional-gradient boosting. We derived the gradients for the M-step by maximizing the lower bound of the gradient and showed how to approximate the E-step. We evaluated the algorithm on three different types of relational learning problems: RDNs, MLNs and imitation learning. Our results indicate that the proposed algorithms outperform the respective algorithms that make closed-world assumptions.

Our approach to handle missing data can also be extended in various directions. Exploring alternative efficient methods for computing the gradients is one such direction. Adapting the different EM heuristics such as random restarts is another interesting direction. We could also calculate the marginal probabilities of each hidden grounding and use them as probabilistic facts to learn the trees. Our approach can handle bidirected and undirected models but extending it to an acyclic directed model is an interesting avenue for future research.

Acknowledgements Tushar Khot, Sriraam Natarajan and Jude Shavlik gratefully acknowledge support of the DARPA Machine Reading Program and DEFT Program under the Air Force Research Laboratory (AFRL) prime contract no. FA8750-09-C-0181 and FA8750-13-2-0039 respectively. Sriraam Natarajan also gratefully acknowledges the support of Army Research Office grant number W911NF-13-1-0432 under the Young Investigator Program. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the DARPA, AFRL, ARO, or the US government. Kristian Kersting was supported by the Fraunhofer ATTRACT fellowship STREAM and by the European Commission under contract number FP7-248258-First-MM.

References

Bellodi E, Riguzzi F (2012) Learning the structure of probabilistic logic programs. In: Inductive Logic Programming (ILP), LNCS, vol 7207

Bellodi E, Riguzzi F (2013) Expectation maximization over binary decision diagrams for probabilistic logic programs. Intelligent Data Analysis 17(2):343–363

Besag J (1975) Statistical Analysis of Non-Lattice Data. The Statistician 24(3):179–195

Biba M, Ferilli S, Esposito F (2008) Structure learning of Markov logic networks through iterated local search. In: European Conference on Artificial Intelligence (ECAI)

Blockeel H, Raedt LD (1998) Top-down induction of first-order logical decision trees. Artificial Intelligence 101:285–297

Chan P, Stolfo S (1998) Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In: Knowledge Discovery and Data Mining (KDD)

Craven M, DiPasquo D, Freitag D, McCallum A, Mitchell T, Nigam K, Slattery S (1998) Learning to extract symbolic knowledge from the World Wide Web. In: Association for the Advancement of Artificial Intelligence Conference (AAAI)

Davis J, Goadrich M (2006) The relationship between Precision-Recall and ROC curves. In: International Conference on Machine Learning (ICML)

Dempster A, Laird N, Rubin D (1977) Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society B.39:1–38

Dietterich T, Ashenfelter A, Bulatov Y (2008) Gradient tree boosting for training conditional random fields. Journal of Machine Learning Research pp 2113–2139

Domingos P, Lowd D (2009) Markov Logic: An Interface Layer for AI. Morgan & Claypool

Friedman J (2001) Greedy function approximation: A gradient boosting machine. Annals of Statistics pp 1189–1232

Friedman N (1998) The Bayesian structural EM algorithm. In: Uncertainty in Artificial Intelligence (UAI)

Getoor L, Taskar B (eds) (2007) Introduction to Statistical Relational Learning. MIT Press

Getoor L, Friedman N, Koller D, Pfeffer A (2001) Learning probabilistic relational models. Relational Data Mining pp 307–338

Gutmann B, Thon I, Raedt LD (2011) Learning the parameters of probabilistic logic programs from interpretations. In: European Conference on Machine Learning (ECML)

Jaeger M (2007) Parameter learning for relational Bayesian networks. In: International Conference on Machine Learning (ICML)

Kameya Y, Sato T (2000) Efficient EM learning with tabulation for parameterized logic programs. In: Computational Logic (CL)
Karwath A, Kersting K, Landwehr N (2008) Boosting relational sequence alignments. In: IEEE International Conference on Data Mining (ICDM)
Kersting K, De Raedt L (2007) Bayesian logic programming: Theory and tool. In: Getoor L, Taskar B (eds) An Introduction to Statistical Relational Learning
Kersting K, Driessens K (2008) Non–parametric policy gradients: A unified treatment of propositional and relational domains. In: International Conference on Machine Learning (ICML)
Kersting K, Raiko T (2005) 'Say EM' for selecting probabilistic models for logical sequences. In: Uncertainty in Artificial Intelligence (UAI)
Kersting K, Ahmadi B, Natarajan S (2009) Counting Belief Propagation. In: Uncertainty in Artificial Intelligence (UAI)
Khot T, Natarajan S, Kersting K, Shavlik J (2011) Learning Markov logic networks via functional gradient boosting. In: IEEE International Conference on Data Mining (ICDM)
Kok S, Domingos P (2009) Learning Markov logic network structure via hypergraph lifting. In: International Conference on Machine Learning (ICML)
Kok S, Domingos P (2010) Learning Markov logic networks using structural motifs. In: International Conference on Machine Learning (ICML)
Kok S, Sumner M, Richardson M, Singla P, Poon H, Lowd D, Wang J, Nath A, Domingos P (2010) The Alchemy system for statistical relational AI. Tech. rep., Department of Computer Science and Engineering, University of Washington, Seattle, WA, http://alchemy.cs.washington.edu
Li X, Zhou Z (2007) Structure learning of probabilistic relational models from incomplete relational data. In: European Conference on Machine Learning (ECML)
Mihalkova L, Mooney R (2007) Bottom-up learning of Markov logic network structure. In: International Conference on Machine Learning (ICML)
Natarajan S, Tadepalli P, Dietterich T, Fern A (2008) Learning first-order probabilistic models with combining rules. Annals of Mathematics and AI 54(1-3):223–256
Natarajan S, Joshi S, Tadepalli P, Kristian K, Shavlik J (2011) Imitation learning in relational domains: A functional-gradient boosting approach. In: International Joint Conference on Artificial Intelligence (IJCAI)
Natarajan S, Khot T, Kersting K, Guttmann B, Shavlik J (2012) Gradient-based boosting for statistical relational learning: The relational dependency network case. Machine Learning 86(1):25–56
Richardson M, Domingos P (2006) Markov logic networks. Machine Learning 62:107–136
Shavlik J, Natarajan S (2009) Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network. In: International Joint Conference on Artificial Intelligence (IJCAI)
Singla P, Domingos P (2008) Lifted first-order belief propagation. In: Association for the Advancement of Artificial Intelligence Conference (AAAI)
Xiang R, Neville J (2008) Pseudolikelihood EM for within-network relational learning. In: IEEE International Conference on Data Mining (ICDM)