To Appear in Machine Learning journal

Gradient-based Boosting for Statistical Relational Learning: The Relational Dependency Network Case

Sriraam Natarajan[#], Tushar Khot, Kristian Kersting[*], Bernd Gutmann[+], Jude Shavlik

[#] Wake Forest University School of Medicine, USA
University of Wisconsin-Madison, USA
[*]Frauhofer IAIS, Germany
[+] K.U Leuven, Belgium

Abstract. Dependency networks approximate a joint probability distribution over multiple random variables as a product of conditional distributions. Relational Dependency Networks (RDNs) are graphical models that extend dependency networks to relational domains. This higher expressivity, however, comes at the expense of a more complex model-selection problem: an unbounded number of relational abstraction levels might need to be explored. Whereas current learning approaches for RDNs learn a single probability tree per random variable, we propose to turn the problem into a series of relational function-approximation problems using gradient-based boosting. In doing so, one can easily induce highly complex features over several iterations and in turn estimate quickly a very expressive model. Our experimental results in several different data sets show that this boosting method results in efficient learning of RDNs when compared to state-of-the-art statistical relational learning approaches.

1 Introduction

Bayesian and Markov networks [38] are among the most important, efficient, and elegant frameworks for representing and reasoning with probabilistic models. They have been applied to many real-world problems such as diagnosis, forecasting, automated vision, sensor fusion, and manufacturing control. Nowadays, the role of structure and relations in the data has become increasingly important: information about one object can help the learning algorithm to reach conclusions about other objects. Therefore, relational probabilistic approaches (also called *Statistical Relational Learning* (SRL)) [18] have been developed which, unlike what is traditionally done in statistical learning, seek to avoid explicit state enumeration as, in principle, is traditionally done in statistical learning through a symbolic representation of states. These models range from directed models [16, 25, 13, 21, 17] to undirected models [12, 2] and sampling-based approaches [43, 42, 39]. The advantage of these models is that they can succinctly represent probabilistic dependencies among the attributes of different related objects leading to a compact representation of learned models.

The compactness and even comprehensibility gained by using relational approaches, however, comes at the expense of a typically much more complex model-selection task: different abstraction levels have to be explored. Recently, there have been some advances in this problem, especially in the case of Markov Logic networks [30, 27, 28]. In spite of these advances, the area of structure learning, although the ultimate goal of SRL, is a relatively unexplored and indeed a particularly hard challenge. It is well known that the problem of learning structure for Bayesian networks is NP-complete [7] and thus, it is clear that learning the structure for relational probabilistic models must be at least as hard as learning the structure of propositional graphical models.

A notable exception in the propositional world is Heckerman et al.'s [20] *directed dependency networks*, which are a collection of regressions or classifications among variables in a domain that can be combined using the machinery of Gibbs sampling to define an approximate joint distribution for that domain. The main advantage is that there are straightforward and computationally efficient algorithms for learning both the structure and probabilities of a dependency network from data. The other advantage is that these models allow for cyclic dependencies that exist among the data and in turn combine to some extent the best of both directed and undirected relational models. Essentially, the algorithm for learning a DN consists of independently performing a probabilistic classification or regression for each variable in the domain. This allowed Neville and Jensen [34] to elegantly extend dependency networks to the relational case (called as *Relational Dependency Networks*) and employ relational probability trees for learning.

The primary difference between Relational Dependency Networks (RDNs) and other directed SRL models such as PRMs[16], BLPs[25], LBNs[13] etc. is that RDNs are essentially an approximate model. They approximate the joint distribution as a product of marginals and do not necessarily result in a coherent joint distribution. As mentioned elsewhere by Heckerman et al. [20], the quality of the approximation depends on the quantity of the data. If there are large amounts of data, the resulting RDN model is less approximate. Neville and Jensen [34] learn RDNs as a set of conditional distributions. Each conditional distribution is represented using a relational probability tree [35] and learning these trees independently is quite effective when compared to learning the entire joint distribution. Therefore, it is not surprising that RDNs have been succesfully applied to several important real-world problems such as entity resolution, collective classification, information extraction, etc.

However, inducing complex features using probability estimation trees relies on the user to predefine such features. Triggered by the intuition that finding many rough rules of thumb of how to change one's probabilistic predictions locally can be a lot easier than finding a single, highly accurate local model, we propose to turn the problem of learning RDNs into a series of relational function approximation problems using gradient-based boosting. Specifically, we propose to apply Friedman's [15] gradient boosting to RDNs. That is, we represent each conditional probability distribution in a dependency network as a weighted sum

of regression models grown in a stage-wise optimization. Instead of representing the conditional distribution for each attribute (or relation) as a single relational probability tree, we propose to use a set of relational regression trees [4]. Such a functional gradient approach has recently been used to efficiently train conditional random fields for labeling (relational) sequences [11, 19] and for aligning relational sequences [23].

The benefits of a boosting approach to RDNs are: First, being a nonparametric approach the number of parameters grows with the number of training episodes. In turn, interactions among random variables are introduced only as needed, so that the potentially large search space is not explicitly considered. Second, such an algorithm is fast and straightforward to implement. Existing off-the-shelf regression learners can be used to deal with propositional, continuous, and relational domains in a unified way. Third, the use of boosting for learning RDNs makes it possible to learn the structure and parameters simultaneously, which is an attractive feature as structure learning in SRL models is computationally quite expensive. Finally, given the success of ensemble methods in machine learning, it can be expected that our method is superior in predictive performance across several different tasks compared to the other relational probabilistic learning methods.

Motivated by the above, we make several key contributions:

- We present an algorithm based on functional-gradient boosting that learns the structure and parameters of the RDN models simultaneously. As explained earlier, this allows for a faster yet effective learning method.
- We compare several SRL models against our proposed approach in several real-world domains and in all of them, our boosting approach equals or outperforms the other SRL methods and needs much less training time and parameter tuning. These real-world problems range over entity resolution, recommendation, information extraction, bio-medical problems, natural language processing, and structure learning across seven different relational data sets.
- Admittedly, we sacrifice comprehensibility for better predictive performance. But, we discuss some methods by which these different regression trees can be combined to a single tree if necessary for human interpretation.
- A minor yet significant contribution of this work is the exploration of relational regression trees for learning RDNs instead of relational probability trees. As we explain, these regression trees allow for a richer representation than the RPTs.

The rest of the paper is organized as follows: in the next section, we review the necessary background. In particular, we outline dependency networks, RPTs, RDNs, functional-gradient boosting method, etc. In the third section, we present the functional-gradient derivation for RDNs. We then present the formal algorithm for boosting RDNs and discuss some of the features and potential enhancements to this learning method. In the experimental section, we provide the results of the learning algorithm when applied in seven different domains. We compare the results of our learning method against state-of-the-art

SRL methods on those problems, demonstrating the robustness of our learning method. Finally, we conclude by discussing some possible future directions for future research.

2 Background

In this section, we present a brief survey of dependency networks (DNs) and their relational extension (RDNs). For a general overview of statistical relational learning, we refer to the book on SRL [18]. We also give a brief introduction to functional gradient boosting in the last subsection.We explain our notations as we introduce them.

2.1 Dependency Networks

Bayesian networks (BNs) [38] have been widely used for learning and reasoning about probabilistic relationships. The graphical structure of BNs encode independencies that occur in the data. While BNs are attractive for modeling, there is a significant drawbacks, in that they cannot capture cyclic dependencies that might exist in the data. Trying to capture the cyclic dependencies in a BN can lead to a denser model than necessary. Also, the problem of inference for arbitrary BNs is NP-hard and as a result, learning BNs is a hard problem (because learning involves the use of repeated inference in the case of missing data). Finally, learning the structure of BNs is extremely hard due to the requirement of enforcing acyclicity conditions.

Heckerman et al.[20], introduced *Dependency networks* (DNs) that approximate the joint distributions of Bayesian networks as a product of individual conditional probability distributions (CPDs). The key advantage of this representation is that these distributions can be learned independently and hence are significantly easier to learn than the general Bayesian networks. However, these gains are not without a price. Since the joint distribution is learned using individual distributions, the resulting model is not necessarily guaranteed to result in a consistent joint distribution. Although this prevents DNs from being used for causal modeling, DNs can be used for encoding predictive relationships. Heckerman et al., [20] prove that a Gibbs sampling method called *ordered pseudo-Gibbs sampler* can be used to recover the full joint distribution, regardless of the consistency of the local CPDs. The only constraints for the proof are that the CPD is positive and the variables of the dependency network are discrete. We refer to the Heckerman et al. [20] for further details on the pseudo-Gibbs sampler.

Graphically, DNs combine the characteristics of both directed and undirected models by allowing bi-directional links between variables. DNs are defined using a bidirected graph $G = (V, E)$. The conditional independencies are interpreted using graph separation as with undirected models [34]. Each vertex v_i corresponds to a feature X_i. Associated with each v_i is a conditional probability distribution $P(v_i|\mathbf{Pa}(v_i))$ that gives the probability of the feature given its parents. Correspondingly, there will be directed edges between the parents $\mathbf{Pa}(\mathbf{v_i})$

and v_i. These edges also encode the conditional independence, $P(v_i|V - v_i) = P(v_i|\mathbf{Pa}(v_i))$. A sample dependency network is shown in Figure 1. As can be seen, there are four random variables $\{A, B, C, D\}$ in the network. This network is very similar to a Bayesian network except for the bi-directional relationship between random variables A and D that breaks the acyclicity. Removal of one of the edges between A and D would yield a Bayesian network or a consistent dependency network where the joint distribution is exactly the product of marginals and not an approximation.

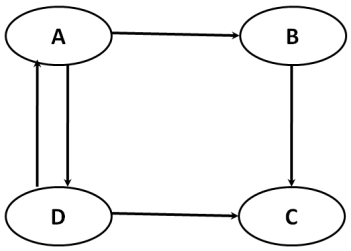


Fig. 1. An example dependency network. Note the bi-directional relationship between random variables A and D.

2.2 Relational Dependency Networks

RDNs [34] extend this idea to the relational world by approximating the joint distribution as a product of conditional distributions over ground atoms. The original formalism of RDNs was motivated using relational databases and the RDN package (Proximity[1]) uses a relational database as its back-end. Since we use a Prolog-based learner in this work, we present RDNs from a logical perspective.

RDNs consist of a set of *predicate* and *function* symbols that can be grounded given the instantiation of the variables. Associated with each predicate Y_i is a conditional probability distribution $P(Y_i|\mathbf{Pa}(Y_i))$ that defines the distribution over the values of Y_i given its parents' values, $\mathbf{Pa}(Y_i)$. We will use capitalized letters (e.g., Y) to denote predicates, small letter (e.g., y) to denote the grounding of the predicate and bold letters to denote the sets of groundings (e.g., given $\mathbf{x}$). Since RDNs are in relational setting, there could be multiple groundings for a predicate. RDNs use aggregators such as *count*, *max* and *average* to combine the values of these groundings.

An example RDN is presented in Figure 2 for an university domain. The ovals indicate predicates, while the dotted boxes represent the objects in the domain. As can be seen, there are *professor*, *student* and *course* objects with *taughtBy* and *takes* as the relations among them. The nodes *avgSGrade* and *avgCGrade*

[1] http://kdl.cs.umass.edu/proximity/index.html

are the aggregator functions over grades on students and courses respectively. The arrows indicate the probabilistic (or possibly deterministic) dependencies between the predicates. For e.g., the predicate *grade* has *difficulty*, *takes*, and *IQ* as its parents. Also note that there is a bidirectional relationship between *satisfaction* and *takes*. As mentioned earlier, associated with each predicate Y_i is a distribution $P(Y_i|\mathbf{Pa}(Y_i))$.

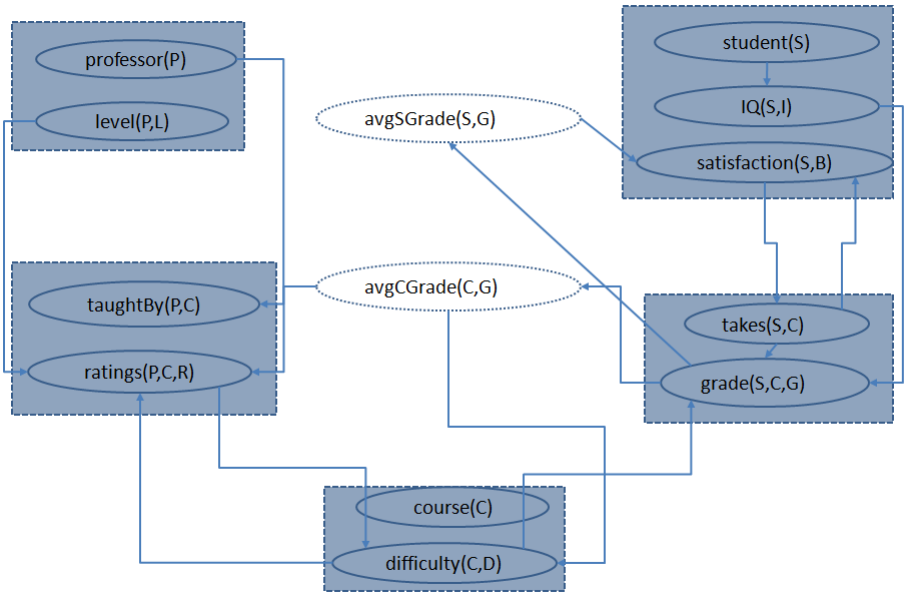


Fig. 2. Example of a RDN.

Learning RDNs: Since the joint distribution of RDNs can be factored into a set of conditional distributions (CPDs), learning RDNs corresponds to learning these distributions. This directly implies that these conditional distributions can each be learned independently of the others. Neville et al. [34] use relational probability trees (RPTs) [35] and relational Bayesian classifiers (RBCs) [36] to capture these distributions. Of the two, RPTs have become a popular method for representing the CPDs in RDNs and hence their RDN learning algorithm learns a RPT for every target predicate P. The RPT tree learner constructs aggregators such as *mode, count, proportion,* and *degree*. The features are restricted to the aggregated predicates. This ensures that there is always one grounding for every path in the tree. (We relax this assumption in this work.) Then feature scores are calculated using chi-square to measure the correlation between the feature and the target.

An example of RPT is presented in Figure 3. This RPT was constructed in [35] for the task of predicting whether a web page is a student web page. This tree

can be interpreted as extending decision trees to the relational setting and thus consists of a series of tests about a web page and its relational neighborhood. The leaves contain the probability distributions over the values of the *isStudent* target label. In this example, the target is Boolean valued. The root node checks if the page is linked to a web page with more than 111 out-links (e.g., a directory page). If so, then the probability of the web page being a student web page is 0.99. Else, the next test is performed which checks whether the page is linked to from a page without a path appended to the URL (e.g., a department home page). If so, it is unlikely to belong to a student as can be seen from the probability being as low as 0.02. If not, the next test is performed. The numbers in the leaf node indicate the number of positive:negative examples that reach the given leaf node. We refer the readers to the work by Neville et al. [35] for further details on RPTs. They demonstrate that RPTs build significantly smaller trees than other conditional models and obtain comparable performance. As we present in the next section, we use relational regression trees instead of RPTs to learn RDNs. The RDNs learned using relational regression trees serve as a baseline to compare against our new approach. This is due to the fact we could not get the RDN software *Proximity* to run on several data sets because of memory issues.

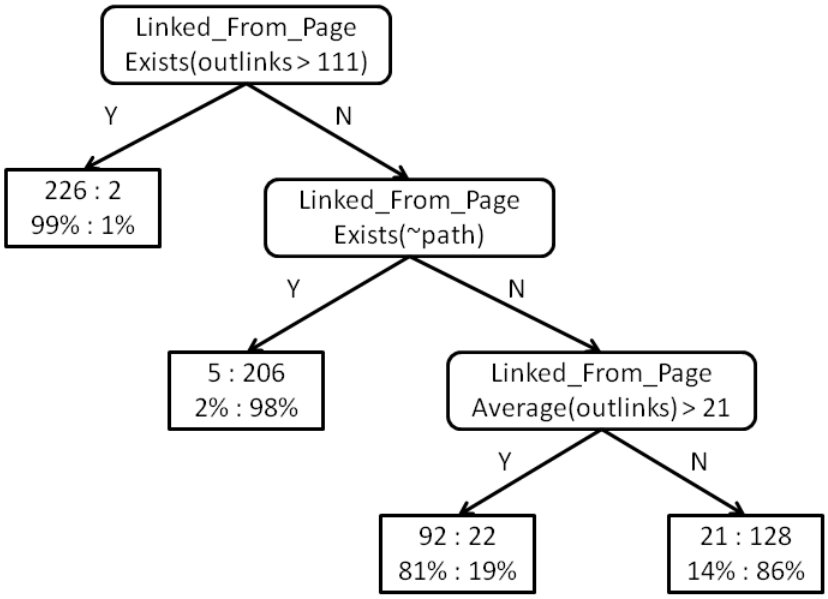


Fig. 3. Example of a RPT (Figure 1 from Neville et al.[35]). The goal of this RPT is to predict whether a web page is a student web page based on the number of outgoing links.Numbers at a leaf node report the number and percentage of positive and negative examples reaching that leaf.

Inference: As with the case of DNs, inference in RDNs is generally performed using *modified ordered pseudo-Gibbs* sampling. To perform inference, Neville et al., unroll the RDN to the ground network (where each predicate is instantiated with all the possible values for the variables in the predicate). Once the ground network is constructed, they perform modified-ordered Gibbs sampling on the unrolled network. Although Gibbs sampling may not be very efficient to estimate the joint distribution, Neville et al. demonstrate that it is reasonable in estimating marginal probabilities for each predicate.

2.3 Functional Gradient Boosting

Functional gradient methods have been used previously to train conditional random fields (CRF) [11] and their relational extensions (TILDE-CRF) [19]. Assume that the training examples are of the form $(\mathbf{x}_i, y_i)$ for $i = 1, ..., N$ and $y_i \in \{1, ..., K\}$. The goal is to fit a model $P(y|\mathbf{x}) \propto e^{\psi(y,\mathbf{x})}$. The standard method of learning in graphical models is based on gradient-descent where the learning algorithm starts with initial parameters θ_0 and computes the gradient of the likelihood function. Note that the parameters of CRFs encode potential functions, i.e., functions mapping configurations on the truth values of some random variables to real values. This view allowed Dietterich et al. [11] to use a different approach to train the potential functions based on Friedman's [15] gradient-tree boosting algorithm where the potential functions are represented by sums of regression trees that are grown stage-wise. Since the stage-wise growth of these regression trees are similar to the Adaboost algorithm [14], Friedman called this *gradient-tree boosting*.

More formally, functional gradient ascent starts with an initial potential ψ_0 and iteratively adds gradients Δ_i. This is to say that after m iterations, the potential is given by

$$\psi_m = \psi_0 + \Delta_1 + ... + \Delta_m \tag{1}$$

Here, Δ_m is the functional gradient at episode m and is given by

$$\Delta_m = \eta_m \times E_{x,y}[\partial / \partial \psi_{m-1} log\, P(y|x; \psi_{m-1})] \tag{2}$$

where η_m is the learning rate. Dietterich et al.[11] suggested evaluating the gradient at every position in every training example and fitting a regression tree to these derived examples. Functional-gradient methods have been used previously to succesfully train conditional random fields (CRF) [11] and their relational extensions (TILDE-CRF) [19], sequence alignments [37] and their relational extensions [23], and policies [47] and their relational extensions [26]. In this paper, we employ the idea of functional gradient boosting to learn relational dependency networks (RDNs). It is worth mentioning that this is the first time that gradient boosting is proposed for relational density estimation.

3 Functional Gradient Boosting of RDNs

As explained in the previous section, *functional gradient* ascent is different from the standard gradient ascent methods in one key aspect - it does not assume a

linear parameterization for the potential function [11]. The standard assumption is that the potential function ψ is represented as

$$\psi = \sum \beta_i f_i \tag{3}$$

where $\{\beta_1, ..., \beta_n\} = \theta$ are the parameters of ψ. As can be seen, the potential function is assumed to be a linear function of the parameters. In functional gradient ascent, ψ is given by Equation 1 where the assumption is more general in that it is a weighted sum of functions and the functional gradient is given by Equation 2. As Dietterich et al. [11] point out, the expectation $E_{x,y}[..]$ cannot be computed as the joint distribution $P(\mathbf{x}, \mathbf{y})$ is unknown.

Since the joint distribution is unknown, functional gradient methods treat the data as a surrogate for the joint distribution. Hence, instead of computing the functional gradient over the potential function, the functional gradients are computed for each training example, i.e.,

$$\Delta_m(y_i; \mathbf{x}_i) = \nabla_\psi \sum_i log(P(y_i|\mathbf{x}_i; \psi))|_{\psi_{m-1}} \tag{4}$$

These are point-wise gradients for each example $(\mathbf{x}_i, y_i)$ conditioned on the potential from the previous iteration(shown as $|_{\psi_{m-1}}$). Now this set of local gradients form a set of training examples for the gradient at stage m. The key step in functional gradient boosting is the fitting of a regression function (typically a regression tree) h_m on the training examples $[(\mathbf{x}_i, y_i), \Delta_m(y_i; \mathbf{x}_i)]$ [15]. Dietterich et al. [11] point out that although the fitted function h_m is not exactly the same as the desired Δ_m, it will point in the same direction (assuming that there are enough training examples). So ascent in the direction of h_m will approximate the true functional gradient.

For the rest of the paper, we drop the predicate notations and denote the query predicates as Y's and the other predicates as X's. Note that when learning a full RDN, each of the predicates become the query predicate successively.

3.1 Derivation of the Functional Gradient

We take a similar approach to learning RDNs. As we have mentioned earlier, an RDN can be represented as a set of conditional distributions: $P(Y|\mathbf{Pa}(Y))$ for all the predicates Y, and learning RDNs corresponds to learning the structure of these distributions along with their values. Functional gradient ascent provides us with solutions to both the problems of structure and parameter learning. We consider the conditional distribution of a variable y_i to be

$$P(y_i|\mathbf{Pa}(y_i)) = \frac{e^{\psi(y_i;\mathbf{x}_i)}}{\sum_{y'} e^{\psi(y';\mathbf{x}_i)}} \tag{5}$$

$\forall x_i \in \mathbf{x}_i$ where $(x_i \neq y_i)$, $\psi(y_i; \mathbf{x}_i)$ is the potential function of y_i given all other $x_i \neq y_i$ and ($y' \in$ all possible groundings of Y_i).

Note that this is a sharp departure from the current setting of RDNs as considered by Neville and Jensen [34]. While they use the notion of relational probability trees to represent the conditional distribution at each predicate, we instead use relational regression trees (RRT) [4] for the same purpose. Though not novel, this is a secondary yet significant contribution of this work. The relational regression trees are more expressive than the RPTs as we explain later in this section. We will also explain the RRT learner later.

Theorem 1. *The functional gradient with respect to* $\psi(y_i = 1; \mathbf{x_i})$ *of the likelihood for each example* $\langle y_i, \mathbf{x}_i \rangle$ *is given by:*

$\frac{\partial \log P(y_i;\mathbf{x_i})}{\partial \psi(y_i=1;\mathbf{x_i})} = I(y_i = 1; \mathbf{x_i}) - P(y_i = 1; \mathbf{x_i})$

where I *is the indicator function that is* 1 *if* $y_i = 1$ *and* 0 *otherwise.*

Proof. The probability of the grounding y_i for the example i is given by

$$P(y_i|\mathbf{x_i}) = \frac{e^{\psi(y_i;\mathbf{x_i})}}{\sum_{y'} e^{\psi(y';\mathbf{x_i})}} \tag{6}$$

Thus, we have

$$\log P(y_i; \mathbf{x_i}) = \psi(y_i; \mathbf{x_i}) - log \sum_{y'} e^{\psi(y';\mathbf{x_i})} \tag{7}$$

Taking the derivative w.r.t the function ψ, we get

$$\begin{aligned}
\frac{\partial \log P(y_i; \mathbf{x_i})}{\partial \psi(y_i = 1; \mathbf{x_i})} &= I(y_i = 1; \mathbf{x_i}) - \frac{1}{\sum_{y'} e^{\psi(y';\mathbf{x_i})}} \frac{\partial \sum_{y'} e^{\psi(y';\mathbf{x_i})}}{\partial \psi(y_i = 1|\mathbf{x_i})} \\
&= I(y_i = 1; x_i) - \frac{e^{\psi(y_i=1;x_i)}}{\sum_{y'} e^{\psi(y';x_i)}} \\
&= I(y_i = 1; \mathbf{x_i}) - P(y_i = 1; \mathbf{x_i}) \square
\end{aligned}$$

Note that the gradient at each example is now simply the adjustment required for the probabilities to match the observed value (y_i) for that example. This gradient serves as the weight for the current regression example at the next training episode.

3.2 Using Relational Regression Trees as Functional Gradients

Following prior work [19], we use *Relational Regression Trees* (RRTs) to fit the gradient function at every feature in the training example. These trees upgrade the attribute-value representation used within classical regression trees. In relational regression trees, the inner nodes (i.e., test nodes) are conjunctions of literals and a variable introduced in some node cannot appear in its right subtree (variables are bound along left-tree paths). Each relational regression tree

can be viewed as defining several new feature combinations, one corresponding to each path from the root to a leaf. The resulting potential functions from all these different relational regression trees still have the form of a linear combination of features but the features can be quite complex [19].

At a fairly high level, the learning of relational regression tree proceeds as follows: The learning algorithm starts with an empty tree and repeatedly searches for the best test for a node according to some splitting criterion such as weighted variance. Next, the examples in the node are split into *success* and *failure* according to the test. For each split, the procedure is recursively applied further obtaining subtrees for the splits. We use weighted variance on the examples as the test criterion. In our method, we use a small depth limit (of at most 3) to terminate the search. In the leaves, the average regression values are computed. We augment the relational regression tree learner with the aggregation functions such as *count, max, average* in the inner nodes thus making it possible to learn complex features for a given target.

An example is presented in Figure 4. The goal in the figure is to predict if A is *advisedBy* B. In the tree, if B is a *professor*, A is not a *professor*, A has more than one publication and more than one publication with B, then the regression value is 0.09. As can be seen for most of the other cases, there are negative values indicating lower probabilities (i.e., the probability of the target given that the particular path is true is < 0.5).

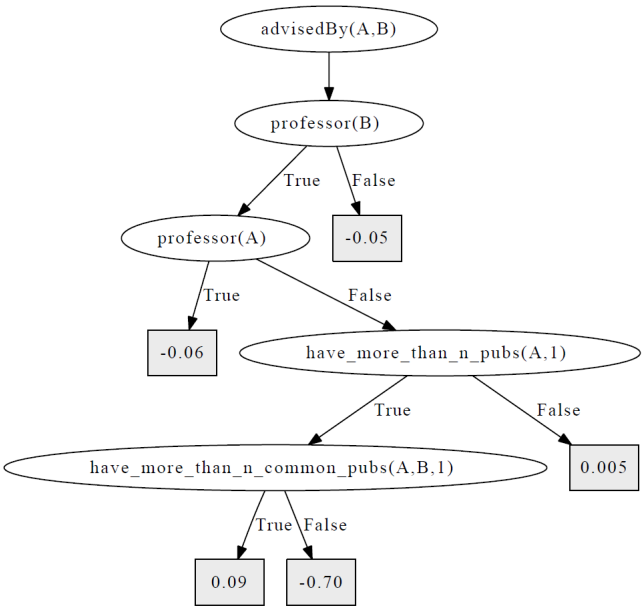


Fig. 4. An Example Relational Regression Tree. The leaves are the regression values for the query advisedBy.

The key features of our regression tree learner are:

- We implemented the version of the regression tree learner *Tilde* [4] that allows for evaluation of nodes containing conjunctions of predicates.
- Our regression tree learner is able to learn recursive rules. To facilitate this, we introduce special predicates (for example, $recursive\_target$ for the *target* predicate) and allow these predicates to be used in the search space. The groundings of these special predicates are created such that when the target is true for a particular grounding, the corresponding recursive predicates are set to true as well. Of course, such an approach will lead to an obvious trivial clause such as, $target(A, B) : -recursive\_target(A, B)$ that is always true. When searching a tree for the current *target*, this clause will always have the highest weight. In order to avoid such a trivial rule, we ensure that when using these recursive predicates, some of the variables that appear in the head do not appear in the same recursive predicate. Only a subset of the variables that appear in the recursive predicates are allowed in the clause. This will allow us to learn rules such as $SameBib(A, C) : - SameBib(A, B)$, $SameBib(B, C)$ since each of the predicates in the body contains only one variable that appears in the head. It should be mentioned that such recursive rules are quite frequently useful in the domains that we consider.
- In our regression tree learner, we can include knowledge to guide the search through the space of trees. We can specify that certain predicates should be considered/not considered. This is quite similar to allowing the background knowledge in many ILP systems such as ALEPH[46] and TILDE[4].
- Our regression tree learner constructs the aggregated predicates on the fly during the search. So when a particular predicate is considered for inclusion in the search space, its corresponding aggregated predicates are considered as well. This approach is a special case to the one presented earlier by Vens et al.[49] for learning complex aggregator functions in relational regression trees.

As explained earlier, the regression tree learner takes examples of the form $[(\mathbf{x}_i, y_i), \Delta_m(y_i; \mathbf{x}_i)]$ and outputs the regression tree h_m that minimizes

$$\sum_i [h_m(y_i; \mathbf{x}_i) - \Delta_m(y_i; \mathbf{x}_i)]^2 \tag{8}$$

over all the examples [11]. Note that in our work, we learn *Relational Regression Trees* as presented by Blockeel and De Raedt [4] and as shown in Figure 4.

Hence, the key idea in this work is to consider the conditional probability distribution of each predicate as a set of relational regression trees. These trees are learned such that at each iteration the new set of regression trees aim to maximize the likelihood of the distributions with respect to the potential function. Hence, when computing $P(a(X)|\mathbf{Pa}(a(X)))$ for a particular value of X (say x), each branch in each tree is considered to determine the branches that are satisfied for that particular grounding (x) and their corresponding regression values are added to the potential ψ.

We developed a regression-tree learner in Java that is similar to the regression tree learner TILDE [4]. This regression-tree learner is built upon an Inductive Logic Programming (ILP) [32] system called WILL (Wisconsin Inductive Logic Learner)[2]. This tree learner requires weighted examples as input where the weight of each example corresponds to the gradient presented above for the corresponding example. Note that the different regression trees provide the structure of the conditional distributions while the regression values at the leaves form the parameters of the distributions. Similar to the relational probability trees [35], we also use aggregators such as *count*, *max* and *average* to handle the case of multiple groundings of a predicate.

It can be easily observed that each path between the root and leaf in the regression tree can be considered as a clause in a logic program. For instance, the left-most path of the tree in Figure 4 is:

$$\texttt{professor(A)} \wedge \texttt{professor(B)} \Rightarrow \texttt{advisedy(A,B)}$$

where the regression value associated with the above clause is -0.06. In our implementation, we maintain a list of these clauses (from left to right). While evaluating a particular query, the clauses are evaluated in the same order (from left to right) and the regression value corresponding to the first satisfied ground clause is returned. Note that since we are in the relational (logical) setting, there can be multiple instances that can satisfied for a particular clause. This is mainly due to the fact that unlike RPTs, we do not include *only* the aggregated variables in the inner nodes of the tree. Instead, all the predicates (not including the current query predicate) are considered along with their aggregated versions (such as *count, mean, max, min, mode*) which allows our tree learner to capture richer relationships among the features. To handle the case of multiple instances for a given clause, once again we use the first satisfied ground clause. This is possible because, if a particular inner node is not an aggregator it is a predicate. In this case, it is interpreted as existential semantics and the first satisfied ground instance is selected.

3.3 Algorithm for Learning RDNs

Our algorithm for learning RDNs using functional gradient boosting is called as *RDN-B* and is presented in Algorithm 1. Algorithm *TreeBoostForRDNs* is the main algorithm that iterates over all predicates. For each predicate (k), it generates the examples for our regression tree learner (that is called using function *FitRelRegressTree*) to get the new regression tree and updates its model (F_m^k). This is repeated upto a pre-set number of iterations M (in our experiments, typically, $M = 20$) or a different stopping criteria (for example, the average change in the gradient value between iterations). Yet another possible criterion could be to stop the growth of the trees if there are $1 - \epsilon$ fraction of examples where the change in gradient is less than δ. We typically set ϵ to be 0.05 and δ to be 0.005. Note that the after m steps, the current model F_m^k will have m

[2] http://www.cs.wisc.edu/machine-learning/shavlik-group/WILL/

Algorithm 1 RDN-Boost: Gradient Tree Boosting for RDN's

1: **function** TREEBOOSTFORRDNS($Data$)
2: **for** $1 \leq k \leq K$ **do** ▷ Iterate through K predicates
3: **for** $1 \leq m \leq M$ **do** ▷ Iterate through M gradient steps
4: $S_k := GenExamples(k; Data; F^k_{m-1})$ ▷ Generate examples
5: $\Delta_m(k) := FitRelRegressTree(S_k; L)$ ▷ Functional gradient
6: $F^k_m := F^k_{m-1} + \Delta_m(k)$ ▷ Update models - Compute set of trees
7: **end for**
8: $P(Y_k = y_k|\mathbf{Pa}(X_k)) \propto \psi^k$ ▷ ψ^k is obtained by grounding F^k_M
9: **end for**
10: **return**
11: **end function**
12: **function** GENEXAMPLES($k, Data, F$)
13: $S := \emptyset$
14: **for** $1 \leq i \leq N_k$ **do** ▷ Iterate over all examples
15: Compute $P(y^i_k = 1|\mathbf{Pa}(x^i_k))$ ▷ Probability of the predicate being true
16: $\Delta(y^i_k; x^i_k) := I(y^i_k = 1) - P(y^i_k = 1|\mathbf{Pa}(x^i_k))$ ▷ Compute Gradient
17: $S := S \cup [(x^i_k, y^i_k), \Delta(y^i_k; x^i_k))]$ ▷ Update relational regression examples
18: **end for**
19: **return** S ▷ Return regression examples
20: **end function**

regression trees each of which approximates the corresponding gradient for the predicate k. These regression trees serve as the individual components ($\Delta_m(k)$) of the final potential function. A key point about our regression trees is that they are not large trees. Generally, in our experiments, we limit the depth of the trees to be 3 and the number of leaves in each tree is restricted to be about 8 (the parameter L in *FitRelRegressTree*). The initial potential F^1_0 is usually set to capture the uniform distribution in all our experiments. However, it is possible to use more informative initial potentials that can encode domain knowledge or the prior about the target.

The function *GenExamples* (line 4) is the function that generates the examples for the regression-tree learner. As can be seen, it takes as input the current predicate index (k), the data, and the current model (F). The function iterates over all the examples and for each example, computes the probability and the gradient. Recall that for computing the probability of y_i, we consider all the trees learned for Y_i. For each tree, we compute the regression values based on the groundings of the current example. The gradient is then set as the weight of the example.

The algorithm *TreeBoostForRDNs* loops over all the predicates and learns the potentials for each predicate. The set of regression trees for each predicate forms the structure of the conditional distribution and the set of leaves form the parameters of the conditional distribution. Thus gradient boosting allows us to learn the structure and parameters of the RDN simultaneously. It should be mentioned that the key component of the learning algorithm is the functional gradient boosting. The relational regression trees serve as one method to capture

these gradients. There could be other ways in which these gradients can be captured. In our setting, the relational regression trees are very natural. This is due to the fact that a set of relational regression trees can replace a single relational probability tree from the original formalism.

3.4 Collective Classification and Gibbs Sampling

The algorithm presented above iterates through each of the predicate and learns the conditional distribution for the corresponding predicate. One of the major advantages of SRL models is the ability to perform collective classification: i.e., reasoning about the queries simultaneously rather than independently. To allow for such an interaction, we include the other query predicates in the training data while learning the model for the current query. This is possible because we assume that in the training data, all the queries are observed.

When some of the predicates are not observed, we use Gibbs sampling for inferring the hidden values based on the current model. The Gibbs sampling method that we use is the same *modified ordered pseudo-Gibbs sampling* method that we have referred earlier. When a particular predicate is not observed for the current example, we sample its value by using the current model for the predicate. We assume that in these cases, there is a natural ordering for the predicates that can be domain-specific. For instance, if we are interested in predicting a particular target, and some of the values of other predicates are missing, those predicates must be sampled before the target is sampled to make them predict the target. When learning a collective classification model, the ordering is not too important: we can re-order the sampling predicates between different sampling iterations and still converge to the same result[34].

Alternatively, it is possible to re-order the steps in the algorithm as follows: instead of learning the complete model for each predicate i.e., learn the set of regression trees for the one query predicate before the others, it is possible to learn the entire model stage-wise. This is to say the lines 2 and 3 of the function $TreeBoostForRDNs$ in the algorithm will be swapped. We loop through the predicates and learn one tree for each predicate. So for each predicate, at the end of each iteration, we will learn a single tree and continue the learning. This will enable the learning of bi-directional relationships that can possibly exist between the query predicates in presence of hidden data. More precisely, if there are more than one target predicate that could be inter-related (for example in collective classification), it is quite possible that some of these predicates are not always observed. In that case, re-ordering the steps will allow the Gibbs sampler to sample a value for the target predicate conditioned on the current model. In most of our experiments, we did not have to make this change to the algorithm as the target predicates were observed in all the data sets.

3.5 Discussion

Note that our boosting algorithm provides a method to learn the structure and parameters of the RDNs simultaneously. The predicates in the different relational

regression trees form the structure of the model (the CPD), while the regression values themselves are the parameters of the CPDs. We believe that the functional gradient boosting might be the solution to the hardest problem in SRL: learning the structure of the model. Note that inference in SRL models is very expensive too, but since inference is the inner loop of the learning algorithms, structure learning is the hardest problem of SRL. Though there are several lifted inference techniques proposed [45, 24, 40, 10, 31] for SRL models, the research in this area is at a nascent stage and boosting would provide the perfect opportunity to learn the structure and parameters simultaneously.

Interpretability of the resulting trees: Having presented the boosting algorithm, the obvious question is: Are we sacrificing *comprehensibility* for better predictive *performance*? Since the trees learned later in the process are dependent on the initial trees, it is certainly true that they cannot be interpreted individually. The entire CPD is a set of regression trees and it does not make sense to interpret them individually. But, this does not necessarily mean that the CPD itself is not interpretable.

We can make the trees interpretable in a few different ways. The first obvious method is to collect all the predicates in the different trees and determine the probability of the target predicate for all the combination of predicates. Once the probabilities are computed, it is possible to imagine the induction of a single tree from the gigantic CPT similar to that of context specific independence trees [5]. Though theoretically possible, this method is quite cumbersome as the number of combinations is exponential in the groundings of the predicates. Instead we use a more practical and a reasonable method. Here the first-step is to predict the probabilities of the target predicate in the training set once all the regression trees are learned. Now the new training data set consists of the training examples along with their probabilities. Hence, it is possible to learn a single tree (without restricting its depth) that would capture the entire data set. This resulting tree can then serve as a surrogate for our set of regression trees. This idea was earlier explored in the context of neural networks by Craven and Shavlik [8]. In this work, we adopt the second method for interpreting the learned trees. An example of the learned tree for predicting if a student is *advisedBy* a Professor is presented in Figure 5.

Bagging: *Bagging* [6] is a technique that is generally employed in machine learning to reduce the variance of the model. As boosting is primarily a bias-reduction technique, there could be non-trivial variance associated with the learned model. To reduce this variance, it is possible to combine the two ensemble techniques of bagging and boosting in a single model. We also tried bagging a set of boosted RDN models, where each run of the RDN-B algorithm used a "boostrap replicate" of the training examples. In addition, to further increase variance across the models we only allowed the RDN-B algorithm, when selecting the best conjunct for a node, to consider a random 50% of the candidate literals, analogous to what is done in decision forests [6, 1]. This approach led to improved predictions,

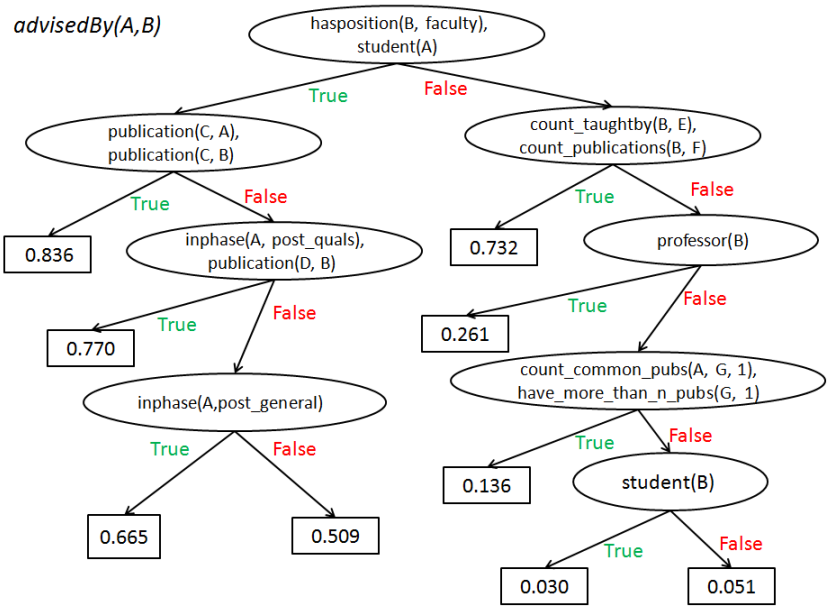


Fig. 5. A single tree induced from the set of regression trees learned for predicting the advisedBy relation between a professor and a student. Note that some of the nodes are conjunctions of predicates with "," denoting the conjunction symbol. The numbers at the leaves are the probability of the advisedBy relation being true.

plus it has the benefit of easy parallelization. It is the bagging+boosting version we use in the NFL data set experiments in the next section. This bagging method reduces the variance for a small number of gradient-steps but as the number of gradient-steps increase,it does not provide significant reduction in variance. This is mainly due to the property of boosting which is a bias-reduction technique for a small number of models and a variance reducing technique as the number of models increases drastically.

4 Experiments

In this section, we present two different kinds of experiments: first is the problem of predicting a single attribute (or relationship) while the second setting is the problem of *collective classification* where the goal is to perform combined classification of a set of interlinked objects. For each of these classification tasks, we use several different relational data sets and several different kinds of problems ranging from entity resolution, to recommendation, to relation extraction. In all these problems, we compare against different SRL methods that have been reported to have the best results in the corresponding data sets. In the cases where we used MLNs, we used the default settings in Alchemy (http://alchemy.cs.washington.edu). Unless otherwise mentioned, we used discriminative learning($-d$ flag) without making any change to the number of it-

erations. We also used the $-queryEvidence$ flag if all the negatives are not enumerated. For inference,we used MC-SAT (option $-ms$) with other flags set to the default values. Also, for all our experiments (except the NFL data set), we subsampled the negatives so that there were twice as many negative as positive examples. This meant that we had to set the initial potential to be -1.8 to capture the uniform distribution. Finally, it should be mentioned that for most of the data sets, we used the setting used in the literature previously and we refer to the previous work as appropriate.

4.1 Prediction of a single relation

We first present our results from three different data sets: (1) UW dataset to predict the *advisedBy* relationship between students and professors (*entity-resolution*); (2) Movie lens dataset to predict the ratings of movies by users (*recommendation*); (3) Predicting adverse-drug reactions to drugs (*bio-medical problem*).

Q1: How does the boosting approach to learning RDNs compare against state of the art SRL approaches on the different kinds of prediction problems?

UW data set: For the UW-data set [12], the goal is to predict the *advisedBy* relationship between a student and a professor. The data set consists of details of professors, students and courses from 5 different sub-areas of computer science (AI, programming languages, theory, system and graphics). Predicates include `professor, student, publication, advisedBy, hasPosition, projectMember, yearsInProgram, courseLevel, taughtBy, teachingAssistant` etc. and equality predicates such as `samePerson, sameCourse` etc. Hence, our task is to learn using the different predicates, to predict the `advisedBy` relation. There are 4,106,841 possible `advisedBy` relations out of which 3380 relations are true.

We trained on four areas and evaluated the results on the other area. This is the same approach taken in the MLN literature [12] where each of the four areas form a "mega-example" that consists of all the inter-related objects of that area. Thus each area can be seen as a single example. Our results are thus averaged over five runs (mega-examples). We compared our RDN-B method against *RDN* (that is learned using a single large regression tree) and MLNs [12]. For *RDN*, we used 20 leaves and the same features that were used for *RDN-B*. For MLNs, we used Alchemy.

The results of the UW-dataset are presented in Table 1. We present the likelihood on the test data ($\sum_i P(y_i = \hat{y}_i)$), the area under curve for PR and ROC and the time taken for training. For computing AUC, we used the code present at http://mark.goadrich.com/programs/AUC/.

As can be seen, *RDN-B* that uses gradient tree boosting has the best likelihood on the test data and is marginally better than *RDN*. As shown in Table 2, both *RDN-B* and *RDN* perform statistically significantly better than MLNs. MLNs were able to identify the negative examples but did not identify the positives well. This fact is also made obvious by the AUC-PR and AUC-ROC curves

Algorithm	Likelihood	AUC-ROC	AUC-PR	Training Time
RDN-B	0.810	0.976 ± 0.01	0.956 ± 0.03	9 s
RDN	0.805	0.894 ± 0.04	0.863 ± 0.06	1 s
MLN	0.731	0.626 ± 0.09	0.629 ± 0.08	93 hrs

Table 1. Results on UW data set. We compare our RDN learner that uses boosting (*RDN-B*), regression learner *RDN* and MLNs. We present the results for the area under curves for ROC and PR, the likelihood of test examples and the training time.

p-value	RDN	MLN
RDN-B	0.0614	0.003
RDN		0.009

Table 2. P-values for two-tailed t-test on AUC-PR for UW-CSE.

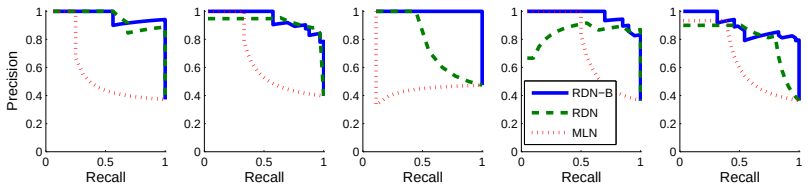


Fig. 6. Precision-Recall curves for UW-CSE data set over 5 folds. Each curve is evaluated on one area which are in the following order: {theory, systems, language, graphics, ai}.

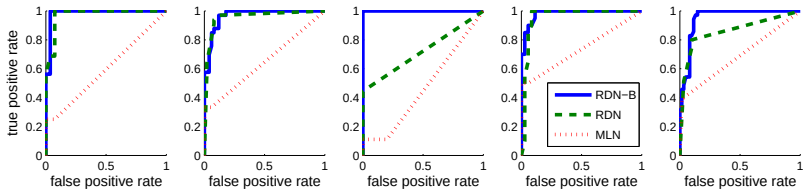


Fig. 7. ROC curves for UW-CSE data set over five folds. Each curve is evaluated on one area which are in the following order: {theory, systems, language, graphics, ai}.

shown for every fold in Figure 6 and 7. Since MLNs are able to identify negative examples well, they are able to get higher precision at the cost of lower recall. For the AUC on both ROC and PR curves, it is clear that *RDN-B* dominates all the other methods. To put this in context, SAYU [9] which had the best reported AUC (for PR curve), is significantly worse than our approach (their reported results were 0.468 for AUC). For MLNs, we had to use all the clauses that predicted *advisedBy* from the Alchemy website since learning the structure on this data set was very expensive. Hence we learned only the weights for these clauses. As can be seen from the last column, weight learning for this data set took us four days as against a few seconds for our approach. As mentioned earlier, we

used the default setting of Alchemy for these runs and it is quite possible that fine tuning the different parameters could yield better results on this problem. We attempted to reduce the running time of Alchemy to five hours but ended up getting poorer results with an average AUC around 0.3 for both ROC and PR curves.

Movie Lens data set: Our next data set is the Movie Lens data set [50]. The dataset was created by randomly selecting a subset of 100 users and 603 movies. The task is to predict the preferences of the users on the movies. The users have attributes `age, gender,` and `occupation` while the movies have released `year` and `genre`. Since we are interested in predicting the preference of the user, we created a new predicate called *likes* for every user-movie combination that takes a value *true* if the user likes the movie and *false* otherwise. Originally, the ratings of the movie by the user were in a 5-star scale. We created the *likes* relationship by setting the value *true* if the rating of a movie by an user is greater than the average rating of all the movies by the same user. Typically, every user rated $(30 - 400)$ movies with a total number of 78,445 user-movie ratings. We performed 5-fold cross validation on the data by choosing 80% of the data to be the training set and evaluating on the other 20%.

Algorithm	AUC-ROC	AUC-PR	Training Time	Accuracy
RDN-B	0.611 ± 0.016	0.602 ± 0.015	332 s	0.587
RDN^*	0.587 ± 0.011	0.587 ± 0.011	6.25 s	0.573

Table 3. Results on Movie Lens data set.

Since this domain involved complex interaction between attributes, we introduced four aggregators for both RDN methods: (a) count of movies rated by the user, (b) count of ratings for a movie, (c) count of ratings of movies of a genre by the user and (d) count of the movies that the user likes in a genre. From Table 3, it can be observed that *RDN-B* is marginally better than *RDN* (statistically significant results in AUC-PR with p-value=0.023). As expected the time taken for boosting is higher when compared to learning a large single tree. We attempted to use Proximity [3] (the default package for RDNs), but ran out of memory. If we restrict the search space, the results were close to random. This is the key reason to use RRTs for learning *RDN* and not the Proximity system. We later present an experiment that compares our method with Proximity. But both the methods are significantly better than MLNs where we used the hyper-graph lifting option of Alchemy to learn the structure [27]. Alchemy was not able to learn any meaningful structure (even with the aggregated predicates) and hence did not learn any useful model. We do not present Alchemy results here as all the examples are predicted *false*.

[3] http://kdl.cs.umass.edu/proximity

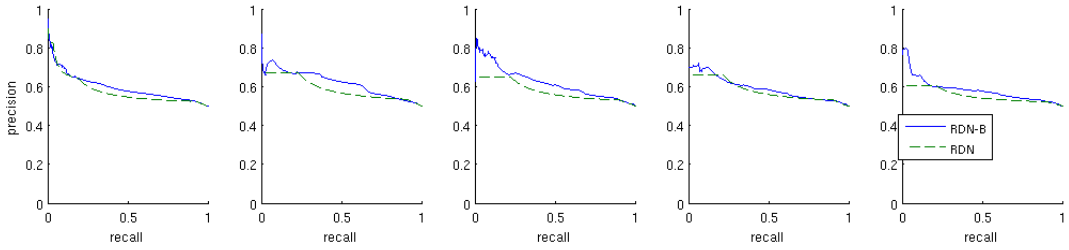


Fig. 8. Precision-Recall curves for Movie Lens data set over 5 folds.

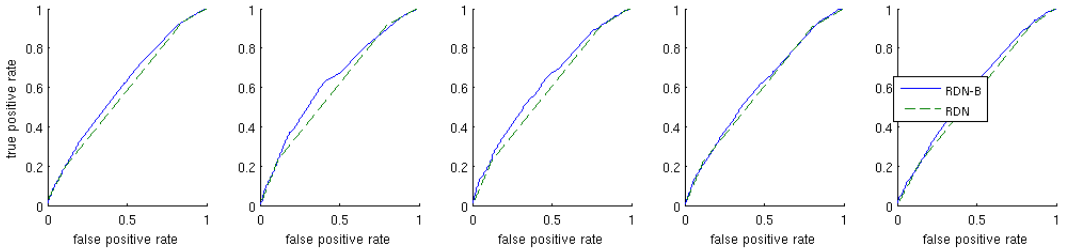


Fig. 9. ROC curves for Movie Lens data set over five folds.

The results of *RDN-B* are quite similar to the best results reported using multi-relational Gaussian Processes [50], where the AUC for ROC was 0.627 for the same experimental setup. The precision-recall and ROC curves for the two RDN-based methods are presented in Figures 8 and 9. As can be seen, there is not much difference between the two methods for this problem. It appears that the aggregators helped both the methods equally. A single tree learned using RDN-B is presented in Figure 10. As can be seen, it uses mostly aggregators such as the count of ratings of the movie, the count of ratings of the user etc. The query in this experiment is $movie\_rating(A, B, C)$ where A is the user, B is the movie and C is the rating. Hence, $count\_ratings\_u\_n(A, C, 50)$ implies user A has rated at least 50 movies as C. $count\_ratings\_m\_n(B, C, 50)$ implies movie B has been rated as C by at least 50 users. This is one problem where boosting did not yield very significant results compared to the non-boosted method of learning RDNs (except for a better predictive accuracy) due the usefulness of these aggregators.

Predicting Adverse Drug Reactions: Our third problem is the prediction of adverse drug reactions on patients. The Observational Medical Outcomes Partnership (OMOP) designed and developed an automated procedure to construct simulated datasets[4] that are modeled after real observational data sources, but contain hypothetical people with fictional drug exposure and health condition

[4] http://omopcup.orwik.com

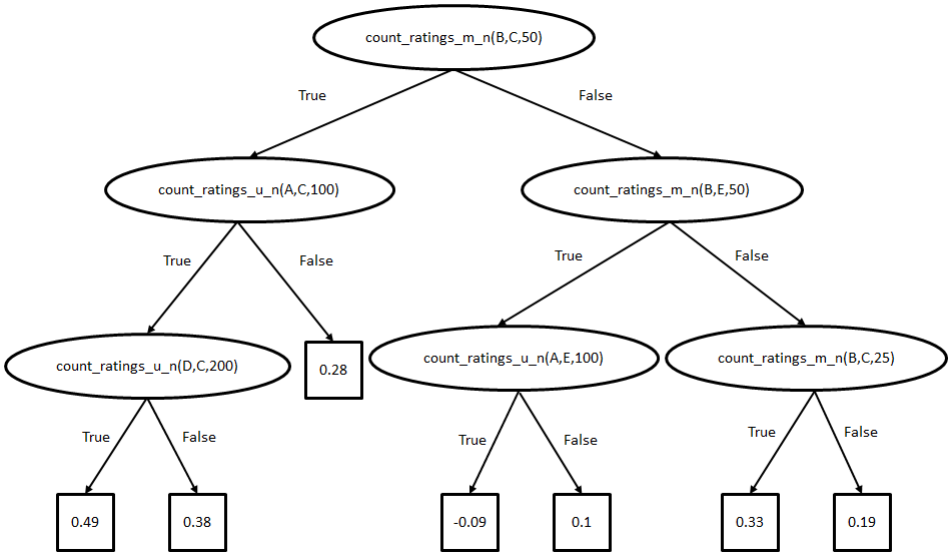


Fig. 10. One regression tree learned by RDN-B for Movie Lens data set.

occurrence. We used the OMOP simulator to generate a dataset of 10,000 patients that included record of drugs and diagnoses (conditions) with dates. The goal is to predict drug use based on the conditions. 75% of the data was used for training, while the remaining 25% was used for testing. The test was conducted on 5 drugs with a training set of **1950** patients on the drug and a test set of **630** patients. We measured accuracy by predicting true if the predicted probability is > 0.5 and false otherwise.

Algorithm	AUC-ROC	AUC-PR	Training Time	Accuracy
RDN-B	0.824 ± 0.04	0.839 ± 0.04	497.8 s	0.753
RDN	0.738 ± 0.04	0.736 ± 0.04	39.4 s	0.697
Noisy-Or*	0.420 ± 0.08	0.582 ± 0.07	-	0.687

Table 4. Results on OMOP data set.

The results are presented in Table 4 with the t-test results shown in Table 5. As can be seen, in this domain our approach *RDN-B* is significantly better than *RDN* in all the metrics except training time. This is due to the fact that in this domain, there were several different weak predictors of the class.

The third row of the table (Noisy-Or) is a relational method where we used Aleph [46] to learn ILP rules. For each drug, we learned 10 rules using Aleph which are essentially Horn clauses with the target in the head. Some examples of the rules are:

```
on_drug(A) :-
   condition_occurrence(B,C,A,D,E,3450,F,G,H).
```

p-value	RDN	Noisy-Or*
RDN-B	0.0021	0.0025
RDN		0.0252

Table 5. P-values for two-tailed t-test on AUC-PR for OMOP dataset.

```
on_drug(A) :-
   condition_occurrence(B,C,A,D,E,140,F,G,H),
   condition_occurrence(I,J,A,K,L,1487,M,N,O).
```

The first rule identifies condition 3450 as interesting while the second rule identifies two other conditions as interesting when predicting whether person A was on the current drug. Note that in pure ILP, the result is the disjunction of these rules (i.e., each rule is evaluated and the resulting concept is true if any of those rules are true). In SRL, we soften these rules using probabilities. Associated with each rule is a conditional distribution $P(head|body)$ and since all the rules have the same head, these rules are combined using the Noisy-Or combining rule. We had to learn two set of parameters for combining the ILP rules using Noisy-Or. For every rule obtained from Aleph, we learn one parameter to capture the conditional probability distribution for the rule being true, $P(head|body)$. We also learned one inhibition probability parameter for the Noisy-Or combining rule. These parameters were learned using the EM algorithm presented in [33] where it is derived for learning the parameters of the distributions and the noise parameters simultaneously. We ran the EM algorithm for 50 iterations. We refer the reader to [33] for further details on the algorithm. Our current approach is significantly better than the ILP-SRL combination in all the evaluation metrics. We were unable to get Alchemy to learn rules for this data set due to the prohibitively large number of groundings in the data.

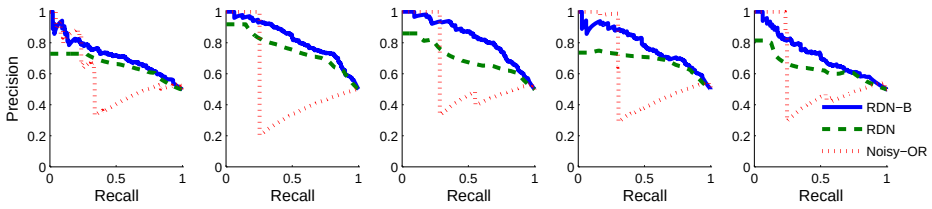


Fig. 11. Precision-Recall curves for predicting drug use of five drugs.

We present the ROC and PR curves for this data set in Figures 11 and 12 respectively. As can be seen, the performance of the RDN-based methods were much better compared to the relational learning method that uses ILP and combining rules. Also, in this data set, boosting greatly helps in performance for almost all the drugs. While the non-boosted RDN is better than the

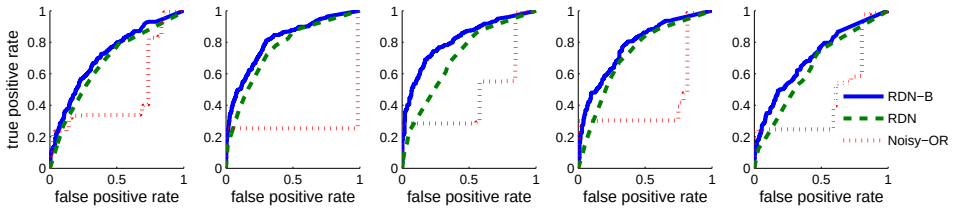


Fig. 12. ROC curves for drug use of five drugs.

ILP/combining rule combination algorithm, RDN-B dominates clearly in this domain.

In summary, we can answer **Q1** affirmatively. Boosting RDN (*RDN-B*) compares favorably (equal to or better than) the state-of-the-art SRL approaches on the three different kinds of tasks. The key benefit is that this performance has been obtained across these domains with minimal parameter tuning.

4.2 Collective Classification

We now present our results of collective classification in four domains: *Internet Movie Database (IMDB)*, *Cora*, *Citeseer* and *National Football League* on different types tasks such as information extraction, structure learning and entity resolution. We reported the running times for the earlier experiments, but on the following experiments, we are not able to report the running times. This is mainly due to the fact that the datasets are large and thus require running on clusters. This meant that different methods may have run on different machines, with various cpu speeds and amounts of RAM; hence, reporting the running times is not meaningful. Thus we avoid the running times and report only the area under curves as had been reported in earlier work on these datasets. The error bars in the bar graphs represent the standard deviations. Also, the NFL dataset was used to understand the relationship to bagging. Since, to our knowledge, not many SRL methods have been applied to this dataset, we compare against the other ensemble method of bagging. For the other datasets, we consider the algorithms that are reported to have the best results. We also present the RDNs learned by boosting for the IMDB and the Citeseer datasets.

Cora dataset: We used the *Cora* dataset for performing two kinds of tasks: *entity resolution* and *information extraction*. Cora dataset, now a standard dataset for citation matching, was first created by Andrew McCallum, later segmented by Bilenko and Mooney [3] and fixed by Poon and Domingos [41][5]. In citation matching, a cluster is a set of citations that refer to the same paper, and a nontrivial cluster contains more than one citation [41]. The Cora dataset has 1295 citations and 134 clusters where almost every citation in Cora belongs to a nontrivial cluster; the largest cluster contains 54 citations. Our experimental setup is similar to the one presented in [41].

[5] Available for download at http://alchemy.cs.washington.edu/papers/-poon07

Q2: How does boosting RDN compare against MLNs on the task of entity resolution?

For the entity resolution task, the following predicates were used: `author, title, venue, sameBib, sameAuthor, sameVenue, sameTitle, hasWordAuthor, hasWordTitle, hasWordVenue`. We make a joint prediction over the predicates - `SameBib, SameTitle, SameVenue` and `SameAuthor`. We used the $B+N+C+T$ MLN presented in Singla et al. [44] and available on the Alchemy website to compare against the boosted and non-boosted versions of RDN. Note that we are **not** learning the structure of MLN, but merely learn the weights of the MLN clauses. In the case of RDN and RDN-B, we learn the structure and parameters of the model.

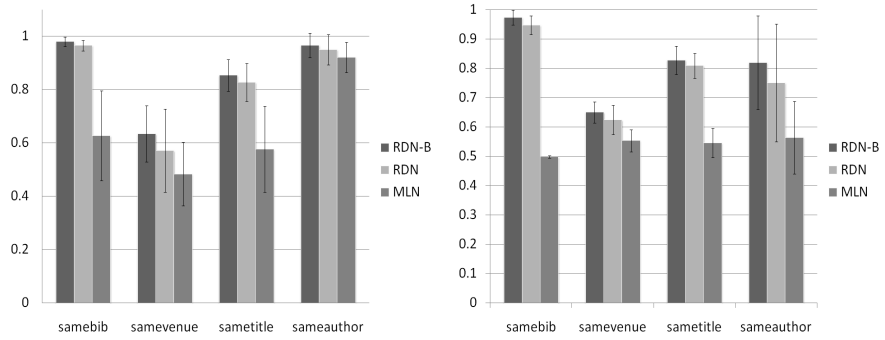


Fig. 13. Area under curves for the entity-resolution task in Cora dataset. The first graph shows the PR curve results while the second one is the area under curve for ROC curves.

The area under curves of the PR curves for the entity resolution task is presented in Figure 13(a). The results are averaged over 5-folds for the four predicates that we mentioned earlier. As can be seen, RDN-B dominates the other methods consistently on all the four different predicates. MLNs exhibit good performance in the case of *SameAuthor* predicate, but are outperformed by the RDN methods in the other predicates. RDN learning that does not use boosting performs reasonably well, but is still outperformed by RDN-B in all the predicates.

Also, the results are similar for the ROC curves and are presented in Figure 13(b). As with the PR curves, these results are obtained over 5-folds. RDN-B dominates for all the predicates in this case as well and is statistically significantly better on SameBib predicate as shown in Table 6.

In summary, we can answer **Q2** positively as we clearly see the superior performance of RDNs learned via boosting compared against the other methods.

Q3: How does boosting RDN compare against MLNs in the task of information extraction?

(a) SameBib

p-value	RDN	MLN
RDN-B	0.037	0.036
RDN		0.037

(b) SameVenue

p-value	RDN	MLN
RDN-B	0.263	0.034
RDN		0.050

(c) SameTitle

p-value	RDN	MLN
RDN-B	0.172	0.033
RDN		0.029

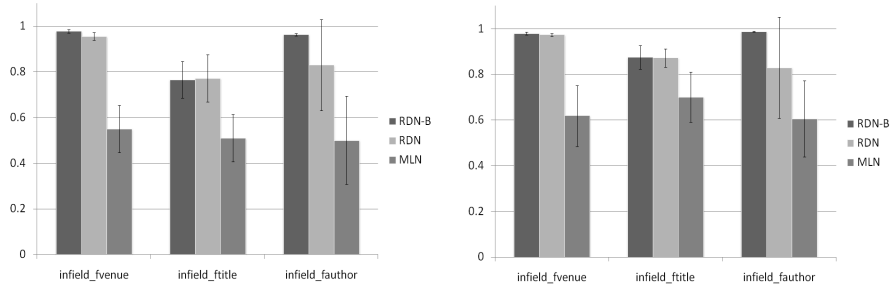
(d) SameAuthor

p-value	RDN	MLN
RDN-B	0.139	0.035
RDN		0.030

Table 6. P-values for two-tailed t-test on AUC-PR for Cora entity-resolution.

For the information extraction task, we use predicates such as the tokens at each position in a citation(`token`), attributes about each token(`isDate, isDigit`, etc.). Given these facts, we try to predict the field type(`Title, Author, Venue`) for each position in the citation. For learning, the field types for each position are known; while for inference, none of the field types are known and need to be inferred jointly. We compared against the MLNs used by Poon et al. [41]. While their work performed entity resolution and information extraction jointly, we only use the features specific to information extraction. Again, we do not learn the structure of MLNs while the RDN-based methods learn both the structure and parameters for this task.

For the boosting method, instead of using two arguments $\langle Bib, Pos \rangle$ in every predicate to indicate a particular position in the citation, we created objects of type *BibPos*. Hence, position P0001 in citation B1000 would result in a *BibPos* object $B1000\_P0001$. Since citations are of varying length, this creation of new objects allows us to avoid predicting labels for positions that do not exist. Also we changed InField(Bib, Field, Pos) to three different predicates: *infield_Field(BibPos)* where *Field*={*Fauthor, Ftitle, Fvenue*}.



(a) AUC-PR (b) AUC-ROC

Fig. 14. Area under curves for the information extraction task in Cora dataset. The first graph shows the PR curve results while the second one is the area under curve for ROC curves.

(a) infield_fvenue

p-value	RDN	MLN
RDN-B	0.183	0.092
RDN		0.097

(b) infield_ftitle

p-value	RDN	MLN
RDN-B	0.958	0.141
RDN		0.110

(c) infield_fauthor

p-value	RDN	MLN
RDN-B	0.355	0.113
RDN		0.182

Table 7. P-values for two-tailed t-test on AUC-PR for Cora information extraction task.

The average area under curves for PR-curves over five folds for the information extraction task are presented in Figure 14(a). We compared RDN-B and RDN against MLNs where we learn the weights using generative weight learning. We could not get discriminative learning of MLN weights working as Alchemy seemed to run out of memory. The results are presented for the three $Infield$ predicates. As can be seen, RDN-B greatly dominates MLNs on all the three predicates. On the other hand, RDNs without boosting are comparable in the *title* field to that of RDN-B. But for the other two fields namely, *author* and *venue*, the performance of RDN-B is significantly better than RDNs without boosting. The results are very similar in the case of ROC curves as well as shown in Figure 14(b). RDN-B dominates MLNs in all the predicates while dominating RDNs significantly in the *author* predicate. The statistical significance results are shown in Table 7.

Citeseer dataset: Similar to the Cora dataset, we used the *Citeseer* dataset created by Poon and Domingos [41] for performing *information extraction*. This dataset was first created by Lawrence et al.[29]. This dataset has 1563 citations and 906 clusters. It consists of four sections, each on a different topic. Over two-thirds of the clusters are singletons and the largest contains 21 citations [41].

Following the methodology used in prior work[41], we created 4-folds using the 4 sections. As with the previous experiment, we compared our learning method against the standard RDN learning and MLNs. We used the MLNs presented in [41]. We learned the weights of the MLN clauses using alchemy and discriminative weight learning setting.

The RDN learned using the boosting algorithm (RDN-B) is presented in Figure 15. The three predicates that are queried jointly in this dataset, viz., *Infield_fauthor, Infield_ftitle,* and *Infield_fvenue* are showed in shaded (dark) ovals. Recall that we created such predicates in the Cora dataset as well. The rest of the predicates are observed in the dataset and hence we do not learn the models for those predicates. This RDN is presented by collecting all the predicates that appear in different trees for a target predicate and making those predicates as the parents of the target predicate. Note that in this dataset, there was no necessity for any aggregation and hence we did not have to create any aggregated features for learning the models.

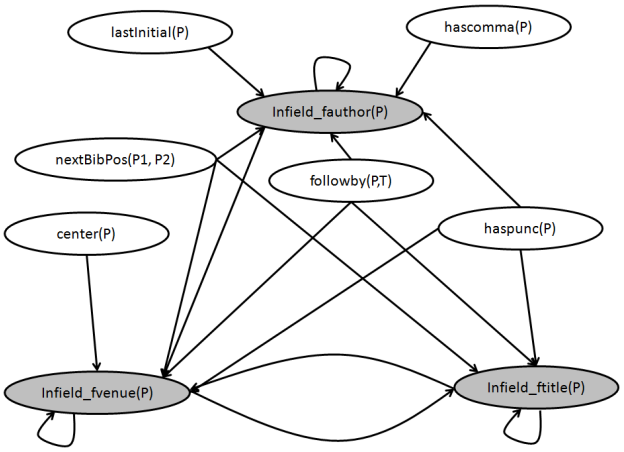
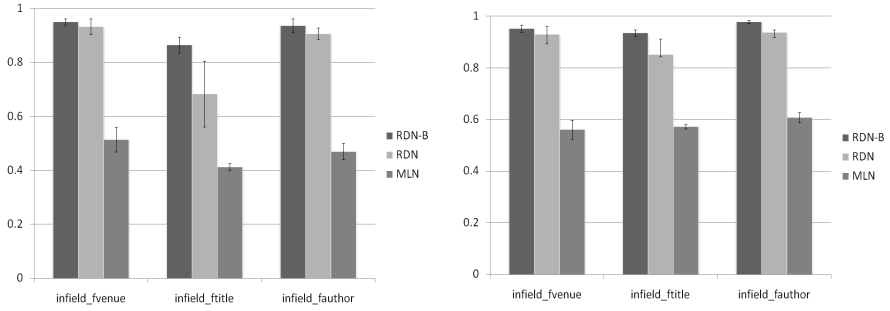


Fig. 15. RDN learned using Boosting for the Citeseer dataset. The nodes that are shaded are the query nodes for which the models have been learned. The set of the nodes that are the parents for the query nodes are the set of all nodes that appear in the different regression trees.

The results of the experiment are presented in Figure 16 with the t-test results shown in Table 8. The experiments were conducted in a similar fashion to the Cora dataset presented before. The bar graphs show that boosted RDN's have a slightly better average AUC for both the ROC and PR curves. MLN's on the other hand have lower average AUC for all the targets.

Based on our results on two different datasets, we can answer **Q3** positively as boosted RDN's outperform most methods on citation segmentation.



(a) AUC-PR (b) AUC-ROC

Fig. 16. Area under curves for the information extraction task in Citeseer dataset. The first graph shows the PR curve results while the second one is the area under curve for ROC curves.

(a) infield_fvenue

p-value	RDN	MLN
RDN-B	0.237	0.037
RDN		0.036

(b) infield_ftitle

p-value	RDN	MLN
RDN-B	0.075	0.036
RDN		0.067

(c) infield_fauthor

p-value	RDN	MLN
RDN-B	0.067	0.035
RDN		0.035

Table 8. P-values for two-tailed t-test on AUC-PR for Citeseer.

IMDB dataset: This dataset that describes a movie domain was created by Mihalkova and Mooney [30] and contains information about actors, movies, directors and the relationships between them. The dataset is divided to 5 independent folds. Following [27], we omitted the 4 equality predicates. The goal is to learn the conditional distribution to predict all the predicates except `actor` and `director`. More precisely, the goal is to perform "structure learning" in this dataset. We compare RDN, RDN-B with two MLN structure learning algorithms BUSL [30] and hypergraph lifting [27]. We used the MLNs from these two papers and did not modify or learn weights for them. We simply queried for the target predicates given the other predicates using these MLNs.

The structure of the RDN learned using boosting (RDN-B) for this task is presented in Figure 17. We collected all the predicates present in the different trees for a particular target predicate and presented them in the figure. The dotted ovals are the aggregated predicates and there are four of them. a_1 is the aggregation performed over two predicates *genre* and *workedUnder*. Similarly, a_2 and a_3 are aggregations performed over *movie* and *genre* and *workedUnder* and *female_gender* predicates respectively. Note that a_4 is the aggregation over a single predicate *workedUnder*. Also, it is worth mentioning that unlike RPTs that allow only for aggregated predicates to be in the model, we allow the original *non-aggregated* predicates as well. We treat the presence of these predicates as existentials, leading to more expressive models compared to original RDN models [34]. It should also be pointed out that when learning RDNs without boosting we allow both aggregated and non-aggregated variables for fair comparison.

Q4: How does the structure learned using boosting RDN compare against the state-of-the-art learning algorithm for learning MLNs?

The area under curve for precision-recall is presented in Figure 18(a). The results are presented for three predicates *worked_under, genre,* and *female_gender*. We do not include predicates such as *actor, director* etc. for evaluation because they can be captured by mutual exclusivity rules and instead focused on the more interesting predicates. We have compared four different algorithms for this task. The algorithms compared are RDN-B that uses boosting, RDN learning that uses a single large regression tree and two different MLN learning algorithms BUSL [30] and hypergraph lifting [27]. The last two algorithms are shown as *BUSL* and *LHL* respectively in the figure. As can be clearly seen, RDN-B performs consistently across all the query predicates. Hyper-graph lifting performs

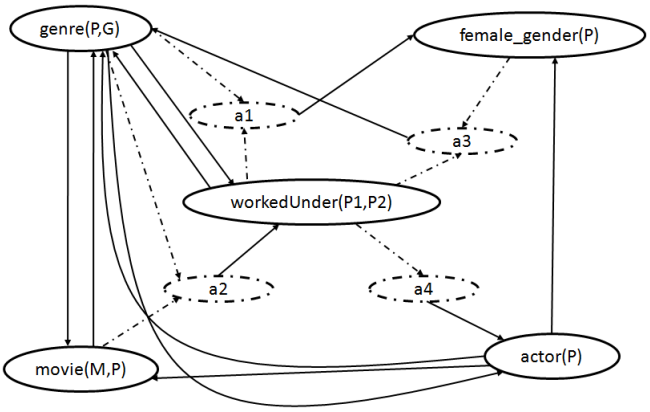
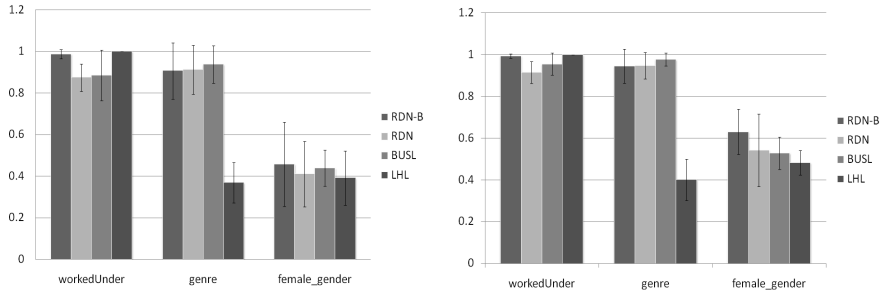


Fig. 17. RDN learned using Boosting for the IMDB dataset. The dashed nodes are the aggregated nodes. For example, a_1 is an aggregation over *workedUnder* and *genre* predicates. For this model, all the nodes form the query predicates.

well for the *worked_under* but BUSL outperforms hyper-graph lifting in the other queries. The statistical significance results are shown in Table 9.



(a) AUC-PR (b) AUC-ROC

Fig. 18. Area under curves in IMDB dataset. The first graph shows the PR curve results while the second one is the area under curve for ROC curves.

The results are quite similar for the ROC curves as well. As can be seen, RDN-B performs consistently well across the different queries. The results for the other methods mirror the results in the PR-curves. Thus **Q4** can be answered affirmatively. RDN-B compares favorably against the state-of-the art methods in the IMDB structure learning task.

NFL dataset: The final dataset that we evaluate our method is the National Football League (NFL) dataset from LDC corpora[6]. This dataset consists of

[6] http://www.ldc.upenn.edu

(a) workedUnder

p-value	RDN	LHL	BUSL
RDN-B	0.039	0.242	0.138
RDN		0.036	0.813
LHL			0.106

(b) female_gender

p-value	RDN	LHL	BUSL
RDN-B	0.534	0.198	0.762
RDN		0.645	0.623
LHL			0.232

(c) genre

p-value	RDN	LHL	BUSL
RDN-B	0.710	0.020	0.508
RDN		0.020	0.426
LHL			0.019

Table 9. P-values for two-tailed t-test on AUC-PR for IMDB.

articles of NFL games over the past two decades. This is essentially a natural language processing (NLP) task. The goal of this experiment is to compare the two different ensemble approaches: bagging and boosting on the NLP task. The idea is to read the texts and identify concepts such as *score*, *team* and *game* in the text. As an easy example, consider the text, "Greenbay defeated Dallas $28 - 14$ in Saturday's Superbowl game". Then, the goal is to identify *Greenbay Packers* and *Dallas Cowboys* as the teams, and 28 and 14 as their respective scores and the game to be a *Superbowl* game. There are cases in which the scores may not be directly specified in the text. The text could specify that "There were 3 touchdowns in the game" and the score must be inferred from this to be 21.

The corpus consists of articles, some of which are annotated with the target concepts. We consider only articles that have annotations of positive examples. The number of positive examples for *score* is 461, while that of *game* is 172 and that of *team* is 780. The number of examples here refer to the number of actual human annotations in the articles. The corresponding negative example counts are 2900, 3300 and $10,700$ respectively. Each article is typically about a particular game, but there could be references to related games. In this case, all the games may or may not be annotated in the article. Hence, there are more annotations for the scores (which may include partial scores) and teams compared to that of the game. The single article might refer to the next team that the winner of the current game would play. This leads to more annotation of the teams Negatives examples for a given target concept were created by randomly sampling from noun phrases not marked as as positive examples of that target concept.

In addition to using the features from the annotated text, we also use the Stanford NLP toolkit[7] to create more features. These features are obtained from the parse trees constructed using the parser, the tags from the part-of-speech tagger, the named entity-recognizer, etc. The features were constructed at different levels: *word-level*, *sentence-level*, *paragraph-level* and *article-level*. Hence,

[7] http://nlp.stanford.edu/software/index.shtml

the data set consisted of the annotations and linguistic information from the NLP parser. These features were provided as inputs to the different learning algorithms.

The methods that are being compared are (1) RDNs (2) Bagging (random forests) (3) Boosted RDNs and (4) Bagging of Boosted RDNs - we touched upon this method in the previous section. The key idea in the *bagging+boosting* method is that we run the boosting algorithm for a few gradient-steps and collect the regression trees and repeat the procedure 100 times and the gradient is averaged over these 100 sets of different trees. Thus the bagging+boosting method is a generalization of our current method that uses a single set of regression trees. We perform five-fold cross validation to predict the above concepts.

Q5: How does boosting RDN compare against bagging and bagging of boosted RDNs in the NLP task?

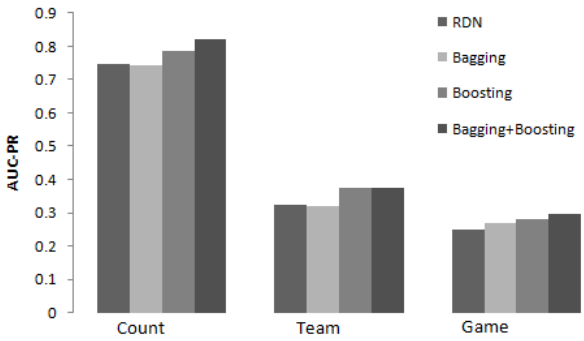


Fig. 19. Precision-Recall values for the NFL corpora. The results are presented for RDNs with a single RRT, Bagged RDNs, Boosted RDNs and Bagged, Boosted RDNs.

The area under curve for precision-recall ($AUCPR$) curves of the different methods are presented in Figure 19. There are three different sets of graphs corresponding to the three concepts: *score(count), game, team.* As can be seen, for all the concepts, the boosted RDN method outperforms standard RDN learning method and the random forests (bagging) method. Interestingly, the method that uses both bagging and boosting helps in one concept (*count*), but is not statistically significantly better in other concepts. It is not clear if bagging always helps and requires more experiments to identify potential benefits of combining the bagging and boosting methods. As can be seen from the figure, while the concept of score is easier to learn, there is a lot of room for improvement for identifying the teams and the game. The key reason is that the natural text is quite ambiguous. For example, one article could mention the team as "San Francisco", the other article could mention them as "49ers" and the third article could be "SF 49ers" etc. Hence, there is a need to perform co-reference resolution across articles and we are currently exploring different methods to do so. We are also currently working on identifying relationships such as *team in a game,*

game winner, game loser, plays in a game etc. We hope to identify the effects of bagging vs. boosting and how bagging can improve boosting etc with these experiments. The NFL experiments reported in this paper are the first-step of the complex NLP task.

4.3 Comparison to Proximity

In most of our earlier experiments, we were unable to compare our boosting algorithm with the original formulation of RDNs. This is due to the fact that the state-of-the-art system for learning RDNs - Proximity ran out of memory for our data sets. Sub-sampling the data might have helped Proximity. But, in all our previous data sets we deal with a single mega example (for example, a single area of research) and hence it is not correct to sample from this mega example.

Hence, in this experiment we compared Proximity to our method on the WebKB data set that is provided as part of the Proximity package. This data is small enough and also has the query model as part of the package. Thus we did not have to make any transformations to the data set. The data set consists of web pages with their most commonly occurring words. It also contains the links between these web pages. The goal is to collectively classify these web pages. While there are multiple categories for the web pages, we present the results of binary classification of web pages into *Student* or *Non-student* pages. We created one fold for every school in the data set (Wisconsin, Washington, Cornell, Texas). This seemed to be a more natural split than an arbitrary sampling and fold creation. This is also consistent with the picture of viewing the set of web pages from a single school as a single mega-example.

Q6: How does the boosting method compare to that of proximity in the binary classification task in the WebKB domain?

We performed 4-fold cross-validation and present the results averaged over these folds in Figure 10. We ignore the predictions made by Proximity on other categories apart from Student to calculate their AUC since we are interested in the binary classification problem. As can be seen from the table, our RDN learning algorithms perform better than Proximity learning algorithm. Table 11 shows the t-test results on the AUC-PR values. We tried increasing the depth of the RPT in Proximity; but it did not improve the results. Between RDN-B and RDNs that use RRTs, there is not a big difference in performance making the results in this domain quite similar to Movie lens data set. We also tried to increase the number of negative examples to see if it makes a big difference. For Proximity, the AUC values decreased marginally for both ROC and PR curves. There was no significant difference in the performance of the RDN algorithms. Hence, providing more negatives did not improve the performance of Proximity.

A part of the relational probability tree (learned using Proximity) and a single relational regression tree (learned using RDN-B) for the WebKB data set are presented in Figures 21 and 20 respectively. Since, the tree learned by Proximity is quite large (32 leaves), we present only a part of the original tree. In this tree, Proximity learns to classify all kinds of web pages (not just student

Algorithm	AUC-ROC	AUC-PR
RDN-B	0.980 ± 0.018	0.965 ± 0.032
RDN	0.980 ± 0.020	0.956 ± 0.050
Proximity	0.753 ± 0.020	0.648 ± 0.028

Table 10. Results on WebKB data set.

p-value	RDN	Proximity
RDN-B	0.723	0.125
RDN		0.100

Table 11. P-values for two-tailed t-test on AUC-PR for WebKB.

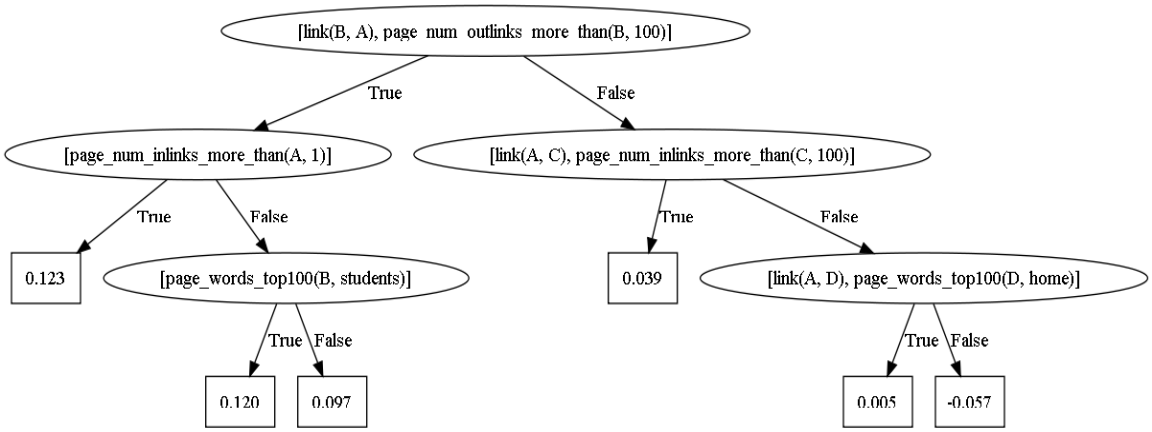


Fig. 20. RRT learned for WebKB data set learned using RDN-B.

or not). The results presented in Table 10 were using 1-Vs-Many classification task for Proximity. We observed that the performance of Proximity is actually worse if we pose the problem as binary classification. Hence, we presented the best results that we could obtain using Proximity.

Discussion In summary, there are a few key questions answered by the experiments. The first is the use of RRTs as against RPTs. In the single experiment that we were able to get Proximity to work, there is a significant difference in performance. A second question is the usefulness of RDNs as against other SRL models. For the cases where we compared to MLNs and other SRL methods, RDNs learned using Boosting demonstrated superior performance. The final and possibly the most important question is the difference between RDNs learned using RRTs and using boosting (RDN vs RDN-B). While in a small number of tasks such as entity resolution problems, RDNs using just a single RRT performed comparably against boosting, in several tasks the performance

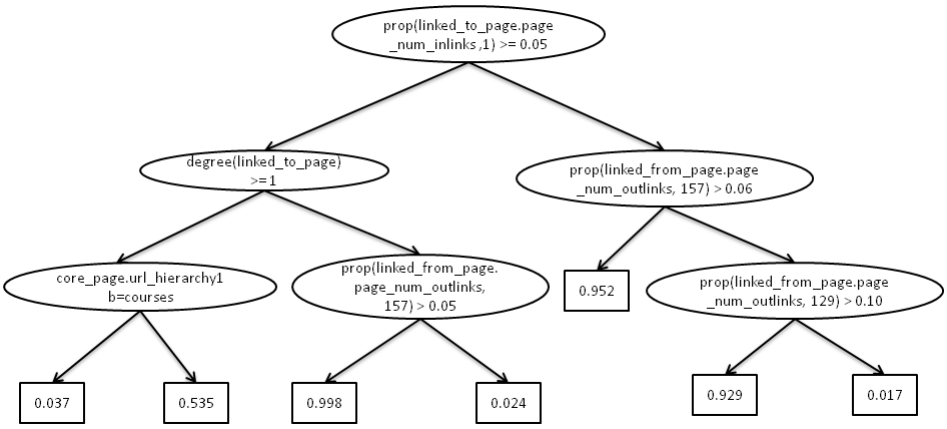


Fig. 21. RPT learned for WebKB data set learned using Proximity.

of RDN-B was clearly superior. An equally important case is that RDN-B was always comparable or better than RRT in all the tasks.

5 Conclusion

Structure learning, that is learning the structure that captures a complex distribution, lies at the heart of statistical relational learning. The higher expressivity of relational models, however, comes at the expense of a more complex structure learning problem: potentially infinitely many relational abstraction levels have to be explored. This may explain why traditional statistical relational learning approaches have not fully achieved their promise yet: they aim at selecting a single model from the data at hand. In general, however, we might need a large number of clauses and this translates to a very large number of parameters.

To account for this, we have presented the first non-parameteric approach to relational density estimation: we turn the problem into a series of relational function approximation problems using gradient-based boosting. For the example of relational dependency networks, we have shown that this non-parametric view on statistical relational learning allows one to efficiently and simultaneously learn both the structure and the parameters of RDNs. We used functional gradient ascent that can be interpreted as boosting regression trees that are grown stage-wise. We demonstrated empirically in several domains that the learning of RDNs using boosting is effective and efficient. While the intermediate structures are not always interpretable, we can always 'compile' the final structures it into 'single tree per predicate' models that are indeed comprehensible. Moreover, the boosting approach yields superior performance over traditional RDNs and other SRL models (such as MLNs [12], SAYU [9] and combining rules based formalisms [33]). Boosting has been previously explored in the context of propositional graphical models [22, 48]. In this paper, we present the first algorithm on boosting for relational probabilistic models.

One possible future direction is to evaluate the approach in several other real-world domains and problems. Another possible research direction is to more thoroughly compare the current approach of gradient-tree boosting against learning random forests (i.e., Bagging). Bagging vs. Boosting has long been an interesting problem in traditional machine learning and it will be worthwhile to compare the methods in the context of relational models (particularly in the case of RDNs). We have taken the first-step in this direction with the NFL dataset as presented in Section 4.2, but more experiments are necessary to draw more useful conclusions. Finally, given that structure learning in SRL models is expensive and relatively unexplored, boosting provides an interesting possibility of structure learning in several different SRL models such as MLNs. Hence, we plan to investigate the problem of boosting for other SRL models in the future.

6 Acknowledgements

We thank the anonymous reviewers for their insightful comments and suggestions that greatly improved the paper. Sriraam Natarajan, Tushar Khot and Jude Shavlik gratefully acknowledge support of DARPA via the Air Force Research Laboratory (AFRL) under contract FA8750-09-C-0181. 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 AFRL. Kristian Kersting was supported by the Fraunhofer ATTRACT fellowship STREAM and by the European Commission under contract number FP7-248258-First-MM. Bernd Gutmann is supported by the Research Foundation-Flanders(FWO-Vlaanderen).

References

1. A. Van Assche, C. Vens, and H. Blockeel. First order random forests: Learning relational classifiers with complex aggregates. In *Machine Learning*, 2006.
2. D. Koller B. Taskar, P. Abeel. Discriminative probabilistic models for relational data. In *UAI*, pages 485–492, 2002.
3. M. Bilenko and R. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, pages 39–48, 2003.
4. H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101:285–297, 1998.
5. C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian Networks. In *UAI*, pages 115–123, 1996.
6. L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.
7. D. Chickering. Learning Bayesian networks is NP-Complete. In *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer-Verlag, 1996.
8. M. Craven and J. Shavlik. Extracting tree-structured representations of trained networks. In *NIPS*, pages 24–30, 1996.
9. J. Davis, I. Ong, J. Struyf, E. Burnside, D. Page, and V.S. Costa. Change of representation for statistical relational learning. In *IJCAI*, 2007.
10. R. de Salvo Braz, E. Amir, and D. Roth. Lifted First Order Probabilistic Inference. In *IJCAI*, pages 1319–1325, 2005.

11. T.G. Dietterich, A. Ashenfelter, and Y. Bulatov. Training conditional random fields via gradient tree boosting. In *ICML*, 2004.
12. P. Domingos and D. Lowd. *MarkovLogic: An Interface Layer for AI.* Morgan & Claypool, San Rafael, CA, 2009.
13. D. Fierens, H. Blockeel, M. Bruynooghe, and J. Ramon. Logical bayesian networks and their relation to other probabilistic Logical models. In *ILP*, 2005.
14. Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *ICML*, 1996.
15. J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, pages 1189–1232.
16. L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning Probabilistic Relational models. *Relational Data Mining, S. Dzeroski and N. Lavrac, Eds.*, pages 307–338.
17. L. Getoor and J. Grant. PRL: A probabilistic relational language. *Machine Learning*, 62(1-2):7–31, 2006.
18. L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning.* MIT Press, 2007.
19. B. Gutmann and K. Kersting. TildeCRF: Conditional Random Fields for Logical sequences. In *ECML*, 2006.
20. D. Heckerman, D. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *JMLR*, 1:49–75, 2001.
21. M. Jaeger. Relational Bayesian networks. In *Proceedings of UAI-97*, 1997.
22. Y. Jing, V. Pavlovi, and J. Rehg. Boosted bayesian network classifiers. *Machine Learning*, 73(2):155–184, 2008.
23. A. Karwath, K. Kersting, and N. Landwehr. Boosting Relational Sequence alignments. In *ICDM*, 2008.
24. K. Kersting, B. Ahmadi, and S. Natarajan. Counting Belief Propagation. In *UAI*, 2009.
25. K. Kersting and L. De Raedt. Bayesian Logic Programming: Theory and Tool. In *An Introduction to Statistical Relational Learning*, 2007.
26. K. Kersting and K. Driessens. Non–parametric policy gradients: A unified treatment of propositional and relational domains. In *ICML*, 2008.
27. S. Kok and P. Domingos. Learning Markov Logic network structure via hypergraph lifting. In *ICML*, 2009.
28. S. Kok and P. Domingos. Learning Markov Logic networks using structural motifs. In *ICML*, 2010.
29. S. Lawrence, C. Giles, and K. Bollacker. Autonomous citation matching. In *AGENTS*, pages 392–393, 1999.
30. L. Mihalkova and R. Mooney. Bottom-up learning of Markov Logic network structure. In *ICML*, pages 625–632, 2007.
31. B. Milch, L. Zettlemoyer, K. Kersting, M. Haimes, and L. Pack Kaelbling. Lifted Probabilistic Inference with Counting Formulas. In *AAAI*, 2008.
32. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.
33. S. Natarajan, P. Tadepalli, T. G Dietterich, and A. Fern. Learning first-order probabilistic models with combining rules. *AMAI*, 2009.
34. J. Neville and D. Jensen. Relational dependency networks. *Introduction to Statistical Relational Learning*, pages 653–692, 2007.
35. J. Neville, D. Jensen, L. Friedland, and M. Hay. Learning Relational Probability trees. In *KDD*, 2003.

36. J. Neville, D. Jensen, and B. Gallagher. Simple estimators for relational bayesian classifiers. In *ICDM*, pages 609–612, 2003.
37. C. Parker, A. Fern, and P. Tadepalli. Gradient boosting for sequence alignment. In *AAAI*, 2006.
38. J. Pearl. *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann Publishers Inc., 1988.
39. D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence, Volume 64, Numbers 1, pages 81-129*, 1993.
40. D. Poole. First-Order Probabilistic Inference. In *IJCAI*, pages 985–991, 2003.
41. H. Poon and P. Domingos. Joint inference in information extraction. In *AAAI*, pages 913–918, 2007.
42. L. De Raedt, A. Kimmig, and H. Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, pages 2468–2473, 2007.
43. T. Sato and Y. Kameya. Parameter learning of Logic Programs for Symbolic-statistical Modeling. *JAIR*, pages 391–454, 2001.
44. P. Singla and P. Domingos. Entity resolution with Markov Logic. In *ICDM*, pages 572–582, 2006.
45. P. Singla and P. Domingos. Lifted First-Order Belief Propagation. In *AAAI*, pages 1094–1099, 2008.
46. A. Srinivasan. *The Aleph Manual*, 2004.
47. R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 2000.
48. T. Truyen, D. Phung, S. Venkatesh, and H. Bui. Adaboost.mrf: Boosted Markov Random Forests and application to multilevel activity recognition. In *CVPR*, pages 1686–1693, 2006.
49. C. Vens, J. Ramon, and H. Blockeel. Refining aggregate conditions in relational learning. In *Knowledge Discovery in Databases: PKDD 2006*, 2006.
50. Z. Xu, K. Kersting, and V. Tresp. Multi–relational learning with Gaussian Processes. In *IJCAI*, 2009.