Boosting Relational Dependency Networks

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

University of Wisconsin-Madison,USA, *Frauhofer IAIS,Germany,
+ K.U Leuven, Belgium

Abstract. Relational Dependency Networks (RDNs) are graphical models that extend dependency networks to relational domains where the joint probability distribution over the variables is approximated as a product of conditional distributions. The current learning algorithms for RDNs use pseudolikelihood techniques to learn probability trees for each variable in order to represent the conditional distribution. We propose the use of gradient tree boosting as applied by Dietterich et al.(2004) to approximate the gradient for each variable. The use of several regression trees, instead of just one, results in an expressive model. Our results in 3 different data sets show that this training method results in efficient learning of RDNs when compared to state-of-the-art approaches to Statistical Relational Learning.

1 Introduction

Bayesian and Markov networks 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 becomes more and more important: information about one object can help the agent to reach conclusions about other objects. Therefore, relational probabilistic approaches [6] have been developed, which seek to avoid explicit state enumeration as, in principle, traditionally done in statistical learning through a symbolic representation of states. 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.

A notable exception is Heckerman et al.'s [8] 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. Essentially, the algorithm consists of independently performing a probabilistic classification or regression for each variable in the domain. This allowed Neville and Jensen [11] to elegantly lift dependency networks to the relational case and employ relational probability trees for learning.

In this paper, we describe a new learning approach for RDNs. Finding many rough rules of thumb of how to change our probabilistic predictions locally can be a lot easier than finding a single, highly accurate local model. Hence, we propose to apply Friedman's [5] 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. Such a functional gradient approach has recently been used to efficiently train conditional random fields for labeling (relational) sequences [3, 7].

The benefits of a boosting approach to RDNs are as follows: First, being a nonparametric approach the number of parameters can grow with the number of training episodes. In turn, interactions among random variables are introduced only as needed, so that the potentially infinite search space is not explicitly considered. Second, it 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 parameter simultaneously which is an attractive feature as structure learning in SRL models is computationally very expensive. Admittedly, we sacrifice comprehensibility for better predictive performance. We compare several SRL models against the proposed approach in three real-world domains and in all of them, our boosting approach outperforms the other SRL methods and needs much less training time.

2 Relational Dependency Networks

Dependency networks [8] approximate the joint distribution over the variables as a product of conditional distributions that can be learned independently. RDNs [11] extend dependency networks to the relational setting. Though RDNs are motivated using relational databases, we present them 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 the predicate, small letter (e.g., y) to denote the grounding of the predicate and bold letters to denote the sets of groundings (e.g., $\mathbf{x}$). 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 of RDN is presented in Figure 1(i) 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 between them. *avgSGrade* and *avgCGrade* 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 *satisfaction* 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 $P(Y_i|\mathbf{Pa}(Y_i))$.

Since RDNs can be represented as a set of conditional distributions, learning RDNs correspond to learning these distributions. Neville et al. [11] use relational probability trees (RPTs) [12] to capture these distributions. We use relational regression trees to learn RDNs that serve as a baseline to compare against our approach. Inference in RDNs is generally performed using Gibbs sampling. Due to space constraints, we skip the details of learning and inference in RPTs and RDNs and refer to [12, 11] for further details.

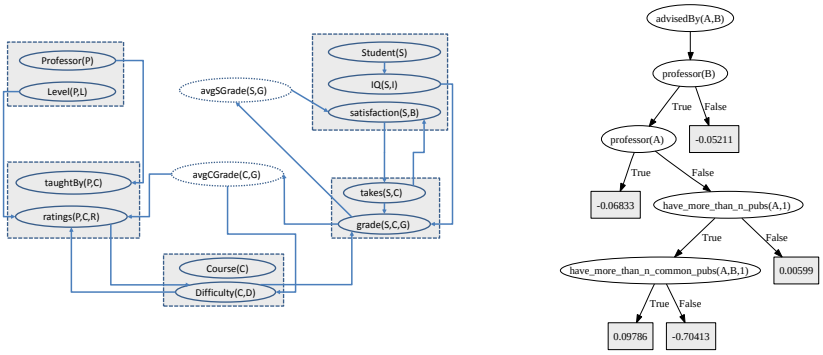

Fig. 1. (i) Example of an RDN (ii) Example of a Relational Regression Tree.

3 Functional Gradient Boosting of RDNs

Functional Gradient methods have been used previously to train conditional random fields (CRF) [3] and its relational extensions (TILDE-CRF) [7]. The standard method of learning in graphical models is based on gradient-descent where the learning algorithm starts with an initial parameter θ_0 and computes the gradient of the likelihood function. For CRFs, when using Functional Gradient Ascent[3, 7], the key idea is to start with an initial potential ψ_0 and iteratively add the gradients Δ. This is to say that after m iterations, the potential is given by $\psi_m = \psi_0 + \Delta_1 + ... + \Delta_m$. Here, Δ_i is the functional gradient at episode i and is given by $\Delta_m = \eta_m \cdot E_{x,y}[\partial/\partial\psi_{m-1} log\, P(y|x; \psi_{m-1})]$. Dietterich et al.[3] suggested the evaluation of the gradient at every position in every training example and fit a regression tree to these derived examples (tree boosting).

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 correspond 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 $P(y_i|\mathbf{Pa}(y_i)) = e^{\psi(y_i;\mathbf{x}_i)} / \sum_{y'} e^{\psi(y';\mathbf{x}_i)}\ \forall x_i \in \mathbf{x}_i \neq y_i$ where $\psi(y_i; \mathbf{x}_i)$ is the potential function of y_i given all other $x_i \neq y_i$. The gradient w.r.t the potential functions is:

$$P(y_i|\mathbf{x_i}) = \frac{e^{\psi(y_i;\mathbf{x_i})}}{\sum_{y'} e^{\psi(y';\mathbf{x_i})}} \Rightarrow logP(y_i|\mathbf{x_i}) = \psi(y_i; \mathbf{x_i}) - log \sum_{y'} e^{\psi(y';\mathbf{x_i})}$$

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

In the above equation I is the indicator function. Note that the gradient is now simply the adjustment required for the probabilities to match the observed value (y_i) in the training data for every example. This gradient serves as the weight for the current regression example at the next training episode. Following prior work[7], we use *relational regression trees* to fit the gradient function at every position in the training example. An example is presented in Figure 1(ii). The goal 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 1 publication and more than 1 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 (< 0.5).

The key idea in our algorithm is to consider the conditional probability distribution of each predicate as a set of 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 and their corresponding regression values are added to the potential ψ. We use aggregators such as *count*, *max* and *average* to handle the case of multiple groundings of a predicate. We use the regression tree learner `TILDE` [1] for learning the regression trees. Due to space constraints, we refer to [1] for a detailed discussion of the tree learner. For this paper, it suffices to mention that the 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.[1]

Our algorithm for learning RDNs using functional gradient is presented in Algorithm 1. Algorithm *TreeBoostForRDNs* is the main algorithm that iterates over all predicates. For each predicate (y_k), it generates the examples for the regression tree learner `TILDE` (that is called using function *FitRelRegressTree*) to get the new regression tree and updates its model (F_m^k). This is repeated for a pre-set number of iterations M (in our experiments, $M = 20$). Note that the after m steps, the current model F_m^k will have m regression trees each of which approximates the corresponding gradient for the predicate y_k. These regression trees serve as the individual components ($\Delta_m(k)$) of the final potential function.

The function *GenExamples* (line 4) is the function that generates the examples for `TILDE`. 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

[1] In reality, the sets of the values at the leaves form the parameters as there could be several possible regression trees for a given predicate.

Algorithm 1 Gradient Tree Boosting for RDN's

function TREEBOOSTFORRDNS($Data$)
  for $1 \leq k \leq K$ **do** ▷ Iterate through K predicates
    for $1 \leq m \leq M$ **do** ▷ Iterate through M gradient steps
      $S_k := GenExamples(k; Data; F^k_{m-1})$ ▷ Generate examples
      $\Delta_m(k) := FitRelRegressTree(S_k; L)$ ▷ Functional gradient
      $F^k_m := F^k_{m-1} + \Delta_m(k)$ ▷ Update Models - Set of regression trees
    end for
    $P(Y_k = y_k|\mathbf{Pa}(Y_k)) \propto \psi^k$ ▷ ψ^k is obtained by grounding F^k_M
  end for
return
end function
function GENEXAMPLES($k, Data, F$)
  $S := \emptyset$
  for $1 \leq i \leq N_k$ **do** ▷ Iterate over all examples
    Compute $P(y^i_k = 1|\mathbf{Pa}(y^i_k))$ ▷ Probability of the predicate being true
    $\Delta(y^i_k) := I(y^i_k = 1) - P(y^i_k = 1|\mathbf{Pa}(y^i_k))$ ▷ Compute Gradient
    $S := S \cup (y^i_k, \Delta(y^i_k))$ ▷ Update relational regression examples
  end for
return S ▷ Return regression examples
end function

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 sets of leaves form the parameters of the conditional distribution. Thus gradient boosting allows us to learn the structure and parameters of the RDN simultaneously.

4 Experiments

We now present our results from: (1) UW dataset to predict the *advisedBy* relationship between students and professors; (2) Movie lens dataset to predict the ratings of movies by users; (3) Predicting adverse-drug reactions to drugs.

Algorithm	Likelihood	AUC-ROC	AUC-PR	Training Time
RDN-B	0.810	0.961	0.930	9 s
RDN^*	0.805	0.888	0.781	1 s
MLN	0.731	0.535	0.621	93 hrs

Table 1. Results on UW data set.

For the UW-dataset [4], the goal was to predict the *advisedBy* relationship between a student and a professor. The dataset consists of professor, student and course details from 5 different sub-areas of computer science. We trained on 4 areas and evaluated the results on the other area. Our results are thus averaged over 5 runs. We compared our method (which we call *RDN-B* to denote RDNs

learned by boosting) against RDN^* (that is learned using a single `TILDE` tree instead of RPTs) and MLNs [4]. For RDN^*, we used 20 leaves and all the same features that were used for *RDN-B*. For MLNs, we used Alchemy[2].

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[3] and the time taken for training. As can be seen, *RDN-B* that use gradient tree boosting has the best likelihood on the test data and is marginally (but statistically significantly) better than RDN^*. MLNs on the other hand were able to identify the negative examples but did not identify the positives well. 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 [2] 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 4 days as against several seconds for our approach.

Algorithm	AUC-ROC	AUC-PR	Training Time	Accuracy
RDN-B	0.625	0.607	332 s	0.587
RDN^*	0.622	0.607	6.25 s	0.573

Table 2. Results on Movie Lens data set.

Our next data set is the Movie Lens data set[14]. We created a randomly selected subset with 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 1 if the user likes the movie and 0 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 1 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. 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%.

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 2, it can be observed that *RDN-B* is marginally better than RDN^* (statistically significant results in accuracy and AUC-ROC). As expected the time taken for boosting is higher when compared to learning a large single tree. We attempted to use Proximity [4] (the default package for RDNs), but ran out of

[2] alchemy.cs.washington.edu

[3] We used http://mark.goadrich.com/programs/AUC/ to compute AUC.

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

memory. If we restrict the search space, the results were close to random. This is the key reason to use `TILDE` for RDN^* and not the Proximity system. But both the methods are significantly better than MLNs where we used the hyper-graph lifting option of Alchemy to learn the structure [9]. 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*. The results of *RDN-B* are quite similar to the best results reported using multi-relational Gaussian Processes [14], where the AUC for ROC was 0.6272 for the same experimental setup.

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

Table 3. Results on OMOP data set.

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[5] that are modeled after real observational data sources, but contain hypothetical people with fictional drug exposure and health outcomes 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.

The results are presented in Table 3. 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. In fact, this is the best result reported so far for this problem. The third row of the table (Noisy-Or) is reported from a recent publication[6] where we used Aleph [13] to learn ILP rules. These rules were then combined using the Noisy-Or combining rule. The parameters were learned using algorithms presented in [10]. 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. We are currently working on experimenting with different settings of Alchemy to learn in this data set. Our experiments conclusively demonstrate that the boosting approach to learning RDNs yields superior performance over other SRL models.

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

[6] Citation of the other work withheld as it is under blind review

5 Conclusion

Structure learning in SRL models is a hard problem. In this work, we propose the use of functional gradient boosting to learn relational dependency networks that allow for efficient learning of both structure and 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 very effective and efficient. While the resulting structure is not always interpretable, the boosting approach yields superior performance over traditional RDNs and other SRL models (such as MLNs [4], SAYU [2] and combining rules based formalisms).

One possible future direction is to evaluate the approach in several other domains. Another possible research direction is to compare the current approach of gradient-tree boosting against learning random forests (i.e., bagging). Bagging vs. boosting has remained 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). Finally, given that structure learning in SRL models is prohibitively expensive, 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 MLNs in the future.

6 Acknowledgements

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. H. Blockeel. Top-down induction of first order logical decision trees. *AI Commun.*, 12(1-2), 1999.
2. J. Davis, I. Ong, J. Struyf, E. Burnside, D. Page, and V.S. Costa. Change of representation for statistical relational learning. In *IJCAI*, 2007.
3. T.G. Dietterich, A. Ashenfelter, and Y. Bulatov. Training conditional random fields via gradient tree boosting. In *ICML*, 2004.
4. P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for AI.* Morgan & Claypool, San Rafael, CA, 2009.
5. J.H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29, 2001.
6. L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning.* MIT Press, 2007.
7. B. Gutmann and K. Kersting. Tildecrf: Conditional random fields for logical sequences. In *ECML*, 2006.

8. 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.
9. S. Kok and P. Domingos. Learning markov logic network structure via hypergraph lifting. In *ICML*, 2009.
10. S. Natarajan, P. Tadepalli, T. G Dietterich, and A. Fern. Learning first-order probabilistic models with combining rules. *AMAI*, 2009.
11. J. Neville and D. Jensen. Relational dependency networks. *Introduction to Statistical Relational Learning*, 2007.
12. J. Neville, D. Jensen, L. Friedland, and M. Hay. Learning relational probability trees. In *KDD*, 2003.
13. A. Srinivasan. *The Aleph Manual*, 2004.
14. Z. Xu, K. Kersting, and V. Tresp. Multi–relational learning with gaussian processes. In *IJCAI*, 2009.