Inductive Logic Programming meets Relational Databases: An Application to Statistical Relational Learning

Marcin Malec, Tushar Khot, James Nagy, Erik Blasch, and Sriraam Natarajan

Abstract

With the increasing amount of relational data, scalable approaches to faithfully model this data have become increasingly important. Statistical Relational Learning (SRL) approaches have been developed to learn in presence of noisy relational data by combining probability theory with first order logic. However most learning approaches for these models do not scale well to large datasets. While advances have been made on using relational databases with SRL models (Niu et al. 2012), they have not been extended to handle the complex model learning (structure learning task). We present a scalable structure learning approach that combines the benefits of relational databases with search strategies that employ rich inductive bias from Inductive Logic Programming. We empirically show the benefits of our approach on boosted structure learning for Markov Logic Networks.

Introduction

Learning of Probabilistic Logic Models and specifically, Markov Logic Networks has received attention lately (Kok and Domingos 2009; 2010). To make learning feasible, some restrictions are assumed including the finite domain assumption (Herbrand interpretations) [1], not allowing for functor symbols, not allowing for recursion etc. In essence, most of these methods mainly exploit "parameter tying" i.e., allowing for instances of objects to share the same parameters under the same set of conditions.

Consequently, many successful systems have been inspired by relational databases (Getoor et al. 2001; Neville and Jensen 2007; Schulte and Qian 2015). For example, more recently, a probabilistic database system called Tuffy (Niu et al. 2012), has been developed for a particular SRL model called Markov Logic network (Domingos and Lowd 2009). However, these systems are restricted to learning only the parameters of the underlying models (weights/probabilities/potential functions) and not the full model (rules/structure of graphical models). This is due to the fact that general rule learning is hard task.

From the logical perspective, Inductive Logic Programming methods (Lavrac and Dzeroski 1994) (ILP) have long exploited background knowledge to constraint the search space while learning restricted (horn) clauses. The background knowledge consists of a set of facts and a small set of rules. These systems learn a hypothesis that combined with the background knowledge can prove maximum number of positive examples while minimizing the number of negative examples proved. Most systems employ additional directives, typically called *modes*, to restrict the search space such that the learning of these clauses is efficient.

We propose to employ the success of ILP methods inside relational databases to accelerate the full model learning of SRL models. Inspired by the recent work on QuickFOIL (Zeng, Patel, and Page 2014), we employ the use of background knowledge inside the database system used by Tuffy. The key difference to QuickFOIL is that we are not just learning a set of rules but a set of weighted rules. To this effect, we adapt the state-of-the-art MLN learning algorithm based on functional-gradient boosting (Khot et al. 2011). This boosting method has been shown to be effectively learning MLNs across several domains and employs the use of modes to guide the search space. Our hypothesis, that we verify empirically, is that combining the scalability of a relational database system with the effectiveness of mode-directed ILP learning will result in huge performance gains compared to the best learning system.

In this work we make the following key contributions: we consider the task of learning SRL models effectively and propose a database solution for this task. We demonstrate how the efficiency and effectiveness of the search space can be improved by using background knowledge inside databases. We consider a powerful learning algorithm and show how it can be further improved by the use of databases. Finally, we demonstrate that the proposed ideas outperform the baseline methods on several benchmark data sets.

Background

We first define some notations that will be used in this work. We use capital letters such as X, Y, and Z to represent random variables (atoms in our formalism). We use small letters such as x, y, and z to represent values taken by the variables and

[1] Some models such as Blog (Milch, Marthi, and Russell 2004) allow for relaxing these assumptions but as far as we are aware, they do not have a full model learning algorithm

bold-faced letters to represent sets.

Markov Logic Networks

A Markov Logic Network consists of a set of formulas in first-order logic and their real-valued weights, $\{(w_i, f_i)\}$. Each grounding of a clause corresponds to a factor with the potential function $\exp(w_i)$, leading to the joint probability distribution, $P(\mathbf{x}) = \frac{1}{Z}\exp\left(\sum_i w_i n_i(\mathbf{x})\right)$, where $n_i(\mathbf{x})$ is the number of times the ith formula is satisfied by $\mathbf{x}$ and Z is the normalization constant. The weights of the rule can be interpreted as weights in Markov networks, i.e. the higher the weights, the more likely is the rule to be true. Due to the exponential nature of the normalization constant, most learning approaches maximize the pseudo-log-likelihood given as $PLL(\mathbf{X} = \mathbf{x}) = \sum_i \log P(X_i = x_i \mid MB(x_i))$ where $MB(x_i)$ is the Markov blanket of x_i.

Boosting MLNs We use the relational functional gradient boosting (RFGB) approach to learn MLNs (Khot et al. 2011). The RFGB approach, similar to Friedman's functional gradient boosting (Friedman 2001), performs gradient ascent on the functional space. To do so, the probability distribution of each relational example, $P(x_i \mid MB(x_i))$ is represented as a sigmoid over a regression function $\psi(x_i; \mathbf{MB}(x_i))$. Given this definition of the probability distribution, the gradients can be computed on the pseudo-log-likelihood function w.r.t. the function ψ as

$$\frac{\partial PLL(\mathbf{X} = \mathbf{x})}{\partial \psi(x_i; \mathbf{MB}(x_i))} = I(x_i = 1) - P(x_i = 1; \mathbf{MB}(x_i))$$

which is the difference between the true distribution (I is the indicator function) and the current predicted distribution. Hence these gradients are positive for positive examples and negative for negative examples.

To perform the gradient ascent procedure, RFGB starts with an initial function ψ_0 defined over all the relational examples (ground atoms) and computes the gradients for all the examples, Δ_1. A regression function, $h_1 : X \to \mathbb{R}$ is then learned to fit to these gradients and added to the initial function i.e. $\psi_1 = \psi_0 + h_1$. This process is repeated n times and the final ψ function for an example is given as the sum of values from all the gradient functions, $\psi_n(x) = \psi_0(x) + h_1(x) + \cdots + h_n(x)$.

Based on the conditional probability distribution of MLNs, the regression function ψ is defined as $\psi(x_i; \mathbf{MB}(x_i)) = \sum_j w_j nt_j(x_i; \mathbf{MB}(x_i))$ where $nt_j(x_i; \mathbf{MB}(x_i))$ corresponds to the non-trivial groundings (Shavlik and Natarajan 2009) of an example x_i given its Markov blanket , $nt_j(x_i; \mathbf{MB}(x_i)) = n_j(x_i = 1, \mathbf{MB}(x_i)) - n_j(x_i = 0, \mathbf{MB}(x_i))$. Relational regression trees or clauses can now be learned to fit to these gradients. In this work, we focus on the learning regression clauses to fit to these gradients. Thus, each gradient step (h_n) is a regression clause and the final model $\psi_n(x) = \psi_0(x) + h_1(x) + \cdots + h_n(x)$ is a sum over the values returned by the regression clauses. Note that learning these clauses would require computing the number of groundings for every candidate clause which can be computed efficiently using databases.

Learning MLNs using Databases

We now present our proposed framework where we employ the use of in-memory databases for learning relational rules with their parameters.

Problem Description

Given: Background knowledge (B), a set of propositional facts – evidence (F), a set of positive (P) and negative examples (N) for a set of target predicates.

To Do: Employ an in-memory database to learn a discriminative MLN using functional-gradient boosting.

Output: The set of learned weighed logic rules (horn clauses).

To this effect, we use the database engine HyperSQL (HSQLDB) in embedded mode. We will consider the following running example throughout the paper.

Illustrative Example: We consider the classic *smokers-friends-cancer* example (Domingos and Lowd 2009). In this problem, we have information about who smokes, and the list of friends relationships. The goal is to learn a model to predict who will have cancer based on their smoking status and the social relationships.

We now present how the background knowledge is encoded in the databases.

Encoding Background knowledge

As mentioned in the background section, the ILP search process takes as input a set of background knowledge and the set of positive and negative examples defined in predicate-logic format. The background knowledge consists of two components:

- Predicate definitions - the names of the predicates and the specification of the domains for the predicate's arguments
- Mode definitions - the rules for the predicate arguments in a candidate literal.

While the first component is standard and have been used widely in several systems, the second mode component is the crucial component for an effective search over a potentially infinite hypothesis space that considers multiple levels of generalization. The modes serve to restrict the language and acts as an inductive bias to the search process. Recall that our current system is inspired from the MLN boosting method (Khot et al. 2011), a discriminative learning approach. The goal is to learn a set of horn clauses and the modes essentially serve to describe the predicates in the hypothesis Horn clauses. An important use of modes is that they serve to restrict the use of existentially quantified variables in the learned horn clauses.

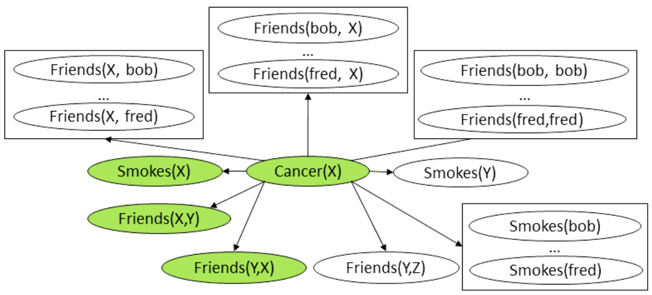
Illustrative Example: Returning to our goal of predicting if a person will have a cancer given his/her relation to other people and the smoking habits of each person, the background file declaration in logic format could look as follows:

```
predDef: friends(person, person).
predDef: smokes(person).
predDef: cancer(person).
mode: friends(+, -).
mode: friends(-, +).
```

```
mode: smokes(+).
mode: cancer(+).
```

It can be observed that the predicates *smokes*, *friends*, and *cancer* all have arguments with domain `person`. Based on the mode specification the new variables considered upon expansion of the current clause would have at least one variable in common with the ones already included in the clause. This is due to the fact that each predicate has a + in the declaration and hence every new predicate that is considered must have the variable at the + location already declared in the clause.

Figure 1: Mode search space reduction.



The use of modes in our learning algorithm can be clearly seen in Figure 1. The current learning task is to predict $Cancer(X)$ (green node in the center). The modes restrict our next expansion search space to the nodes shown in green. As can be seen due to the use of + in Smokes predicate, we only consider $Smokes(X)$ for expansion and not a new existential variable say $Smokes(Y)$. Similarly, some of the friends of X must be introduced into the search space before considering their friends and their smoking habits. These constraints are key for ILP systems to work efficiently and we adapt them in the context of learning with databases.

Facts

Given that the background knowledge is specified as mentioned above, we now show how the facts and the positive and negative examples are encoded in our work. Following prior work in SRL, we make the *closed-world assumption*, i.e., all the groundings that are not specified in the fact base (unobserved groundings) are false.

All the true facts are stored in the database with each predicate corresponding to one table and each argument of the predicate corresponding to a column in the table [2].

In the case of target predicates we use an additional column that contains the truth value of the grounding. Since we are learning a MLN, the MLN semantics requires us compute *PSUM* ($\Sigma_i SATcount_i(x) \times clauseWeight_i$) for each example which is stored as an additional column. This is essentially a sum over the weighted count of the number of satisfied groundings of each clause. Recall that we are performing functional gradient descent, and hence we also need to compute the gradients (*Truth-value* $-$ *sigmoid*(*PSUM*)) for each example. Finally, given the need to compute the difference between the number of satisfied and unsatisfied groundings in the gradient, we also store the negative facts. In our experiments, *PSUM* is initialized to -1.8 (as an initial prior as it was suggested in the work of Khot et al (2011)). In the next section, we show how the facts and background knowledge of the smokers example is fully encoded in our database.

Illustrative Example: Let us consider the task of predicting cancer. Let the true facts for this domain be as follows:

smokes(chuck)	friends(bob, chuck)	cancer(bob)
smokes(bob)	friends(bob, dan)	cancer(chuck)
	friends(chuck, bob)	cancer(fred))
	friends(chuck, fred)	
	friends(dan, bob)	
	friends(fred, chuck)	

These facts would be stored inside the database as following:

atom_Cancer

Truth	PSUM	G	ARG0
1	-1.800	0.858	bob
1	-1.800	0.858	chuck
1	-1.800	0.858	fred
0	-1.800	-0.142	dan

As can be seen above, the groundings of the *Cancer* predicate (which is the query predicate) are stored as a table with the log priors given as PSUM. The gradients are essentially the initial values based on the priors and these are stored in the table as well. They will be modified through the learning process with the aim of driving them to 0.

Given that the positive and negative examples are stored as tables, now the rest of the facts are captured using the friends and smokes tables. They are presented below.

atom_Friends

ARG0	ARG1
bob	chuck
bob	dan
chuck	bob
chuck	fred
dan	bob
fred	chuck

atom_Smokes

ARG0
chuck
bob

[2]We typically store integer ids instead of text strings conserve memory, however, this is not crucial to our approach. Note that we create a one-one mapping between the strings and the ids.

Finally, the gradient G is computed using the following query.

```
Update atom_Cancer SET G = truth - (1.0
/ (1.0 + exp(-PSUM)))
```

As can be seen, this is the initial value of the gradient which is computed using the truth value (1 for true and 0 for false grounding) and the prior weight (PSUM). Given these declaration of the background and the facts, we now turn our attention to implementing the ILP search.

ILP search using databases

The search begins with a horn clause with head being the target predicate. The database representation of the initial clause would consist of a view K that corresponds to the groundings of the initial clause with column names changed to variables.

The next step is to calculate the score of the clause. This is one of the steps where querying a database can be extremely useful. First, we filter out clauses that cover too many or too few examples as they would be not discriminative. In our experiments, we filtered clauses that covered or ignored 97.5% of the examples. For the accepted clauses, a table I is created which contains positive satisfiability counts for the groundings of the head atom. The entries in the table are populated using the following query:

```
Select count(*), head's vars group by
head's vars
```

To compute the weight we would join the I table with the target table to link the gradient values, and then do the computation using aggregate functions:

```
weight = Select sum(G * SAT) / sum(SAT
* SAT) FROM I inner join atom_target on
var1 = arg0 ...
```

The next step would be to compute the score using an outer join in the following manner:

```
score =- Select sum(Power((SAT * weight
- G), 2)) FROM I right outer join
atom_target on var1 = arg0...
```

Illustrative Example: Returning to the the task of modeling cancer, to expand the initial clause to include Smokes(X), we use the following queries:

```
Entries in I table:
Select count(*), var1 group by var1
```

```
weight = Select sum(G * SAT) / sum(SAT
* SAT) FROM I inner join atom_cancer on
var1 = arg0
```

```
score =- Select sum(Power((SAT * weight
- G), 2)) FROM I right outer join
atom_cancer on var1 = arg0
```

The entries in the I table are then:

I table

SAT	var1
1	bob
1	chuck

This process would be repeated for every candidate literal, and then for each of the resulting clauses limited using beam search. The best clause found using such search would then be added to the model. Once a clause is added to the model its I table's SAT counts and clausal weight are used to update the PSUM values of the head's atom table. Following that the gradient values are recomputed.

Use of Modes: To reduce the time spent on doing exhaustive ILP search, modes are used to prune the candidate literals. To generate this reduced set of candidate literals all combination of atoms are generated with restriction that domain of each predicate argument is limited to existing variable if + is specified, and existing variable and possible new variables if - specified, or constants if # is specified. These are stored in a set to eliminate duplicates. For the cancer task, the candidate literals considered in the first gradient step would only include $\langle Smokers(X), Friends(X,Y), Friends(Y,X)\rangle$.

To speed-up the search each gradient step is limited to expanding only 10 best clauses in each gradient step. Finally, the SAT counts do remain the same across gradient iterations, so the I tables are not reused if the same clause is to be evaluated again.

The conversion to the database format allows for efficient query and retrieval of the data. This in turn allows for counting the satisfied groundings of any clause efficiently. As has been shown before(Poyrekar, Natarajan, and Kersting 2014), counting the satisfied grounding is the bottleneck in many PLM tasks including learning and inference. Efficient grounding could possible allow for improving the speed of these tasks.

It must be mentioned that our efficiency does have some limitations - (1) we assume a finite set of groundings (possibly a large set but a finite set). (2) Only horn clauses can be learned using our method and (3) We make the closed-world assumption to perform counting efficiently. However, we argue and show empirically that these assumptions are practically useful in many PLMs. Particularly, the state-of-the-art learning method for MLNs make these assumptions but is built on a logic-based system. We replace the logic based system with our database system and show significant efficiency gains without losing the performance accuracy.

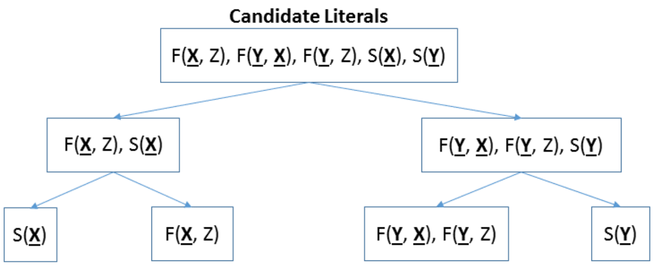
Partitioning Candidate Literals

We partition the candidate literals into groups in which members of the same group share a common join. The idea is to do the shared join only once to speed up the learning time. An example of partitioning is shown in Figure 2.

Application of the system for learning MLNs

Algorithm 1 describes our approach applied to boosting MLNs (Khot et al. 2011). `MLN Boost` function presents the boosting approach as described by Khot et al. (2011). We first generate the regression examples

Figure 2: Use of partitions.




```
Function MLN Boost (Data)
  for 1 ≤ m ≤ M do
    F_m := F_{m-1}
    for P in T do
      S := GenExamples(Data; F_{m-1}, P)
      Δm :=
       FitRelRegressClauseDB(S, P, N, B)
      F_m := F_m + Δm
    end
  end
Function FitRelRegressionClauseDB ((S, P, N,
 B))
  Beam := {P(X)}
  BC := P(X)
  while ¬ empty(Beam) do
    Clause := popFront(Beam)
    if length(Clause) ≥ N then
      continue
    end
    C := getCandidateLiterals(Clause)
    Q := getPartitions(C)
    QCounts =
    getCountsUsingJoins(Q, Clause)
    CCounts :=
    evaluateClauses(P, C, Counts)
    for c ∈ C do
      c.score = SE(c, CCounts(c), S)
      if c.score ≥ Clause.score then
        insert(Beam, c, c.score)
      end
      if c.score ≥ BC.score then
        BC := c
      end
    end
    while length(Beam) ≥ B do
      popBack(Beam)
    end
  end
  return BC
```

Algorithm 1: MLN-Boost Algorithm

based on the gradients described earlier and learn regression clauses to fit these gradients. We change the regression clause learner to use our database representation in `FitRelRegressionClauseDB`.

We use the standard beam search to search over the space of candidate clauses. The parameter N specifies the maximum length of the learned clauses (set to 3 in our experiments) and B specifies the beam size (set to 10). To compute the score of the candidate literals, we first compute the partitions of the literals being considered in `getPartitions`. We use database queries to get the counts of the partitions joined with the current clause in `getCountsUsingJoins`. Finally given these counts over the partitions, we can compute the counts of each example for every candidate literal (`evaluateClauses`). These counts can then be used to compute the squared error (`SE`) while scoring the literals during the search process.

Empirical Evaluation

We now present the experimental results using our approach on standard benchmark SRL data sets. Specifically, the goal in this section is to explicitly evaluate the following questions:

- **(Q1)** Does the proposed database based SRL system outperform the baseline in terms of learning time?
- **(Q2)** Does the proposed system sacrifice learning performance for efficiency gains?

Since we are in relational domains, it is well-known that most of the relations are false - i.e., negative examples far outnumber the number of positives. In such cases, it has been frequently observed that other measures such as Area under the Precision-Recall curve (AUC-PR), Area under Receiver Operating Characteristic curve (AUC-ROC) are considered more reasonable alternatives. Hence, we primarily focus on three performance measures - AUC-ROC and AUC-PR for measuring the performance efficacy and the time in seconds for measuring efficiency. Further, for Cora, IMDB and WebKB datasets we have sub-sampled the negative examples at each gradient step during learning to be twice in number as the number of the positive examples. Our hypothesis is that our system can match the state-of-the-art learning algorithm in learning an accurate model in significantly faster time. To verify the hypothesis, we consider the following approaches:

1. BoostR - WILL based MLN boost algorithm, that serves as our reliable baseline.
2. DB_Boost_NM - Database powered MLN boost without modes, that serves as our DB baseline. This system searches exhaustively for the horn clauses.
3. DB_Boost - Database powered MLN boost that caches join results.

Smokers

	AUC-ROC	AUC-PR	Time(s)
BoostR	1.0	1.0	2.002
DB_Boost_NM	0.5	0.6	2.196
DB_Boost	1.0	1.0	0.376

Smokers is a popular synthetic test bed that is used by several SRL methods for evaluation (Domingos and Lowd 2009; Khot et al. 2011; Natarajan et al. 2012). It consists of

3 predicates: Smokes, Friends, and Cancer. We chose cancer to be our target, our train domain had 6 people, and our test domain had 8 people.

Being a small domain, we do not expect significant improvement in run times. However, as can be observed, the database boosting method that uses modes is still thrice as fast as the baseline method with the same AUC.

Cora Entity Resolution

	AUC-ROC	AUC-PR	Time(s)
BoostR	0.521	0.141	804.877
DB_Boost_NM	-	-	>7200
DB_Boost	0.511	0.157	13.030

The Cora dataset, now a standard dataset for citation matching, was first created by Andrew McCallum, later segmented by Bilenko and Mooney (Bilenko and Mooney 2003), and fixed by Poon and Domingos (Poon and Domingos 2007). In citation matching, a group is a set of citations that refer to the same paper, and a nontrivial group contains more than one citation (Poon and Domingos 2007). The Cora dataset has $1,295$ citations and 134 groups where almost every citation in Cora belongs to a nontrivial group; the largest group contains 54 citations. It contains the predicates: HasWordAuthor, HasWordTitle, HasWordVenue, Title, Venue, Author.

We performed 5-fold cross-validation, and we record average time over the 5 folds. Without the use of modes the database boost algorithm search was not making much progress and we have terminated it at 2 hours. As with the previous experiments, it can be observed that the learned models of our approach exhibit the same prediction performance with databases as that of the original BoostR system. This answers Q2 by showing that we do not sacrifice learning performance while still being significantly faster than the original system.

IMDB

	AUC-ROC	AUC-PR	Time(s)
BoostR	0.986	0.527	27.741
DB_Boost_NM	0.508	0.147	4525.743
DB_Boost	0.985	0.513	3.432

The IMDB dataset was first used by Mihalkova and Mooney (2007) and contains five predicates: actor, director, genre, gender and workedUnder. Since gender can take only two values, we convert the gender(person,gender) predicate to a single argument predicate `female_gender(person)`. Following prior work (Khot et al. 2011), we omitted the four equality predicates. We performed five-fold cross-validation using the folds generated by Mihalkova and Mooney to build model for the target workedUnder and we record average time over the 5 folds.

In this data set, both systems achieve comparable AUC-ROC. However, the database based system seem to have a significantly higher AUC-PR. This is due to improved recall. Investigating the cause of this improvement is an important research direction. In terms of learning time, both systems are fast. However, the proposed system is still significantly faster than the original BoostR system.

WebKB

	AUC-ROC	AUC-PR	Time(s)
BoostR	0.932	0.038	4.161
DB_Boost_NM	-	-	>7200
DB_Boost	0.936	0.039	1.221

The WebKB dataset was first created by Craven et al. (Craven et al. 1998) and contains information about department webpages and the links between them. It also contains the categories for each webpage and the words within each page. This dataset was converted by Mihalkova and Mooney (2007) to contain only the category of each webpage and links between these pages. They created the following predicates: Student(A), Faculty(A), CourseTA(C, A), CourseProf(C,A), Project(P, A) and SamePerson(A, B) from these webpages. The textual information was ignored. We removed the SamePerson(A, B) predicate as it only had groundings with both the arguments being exactly same (i.e., SamePerson(A,A)). We evaluated our method over the CourseProf predicate. We performed 4-fold cross-validation where each fold corresponds to one university, and we record average time over the 4 folds. Without the use of modes the database boost algorithm search was not making much progress and we have terminated it at 2 hours.

As with the previous experiments, it can be observed that the AUC-ROC and AUC-PR are comparable with the BoostR system for the different database systems. However, the proposed system is significantly faster than the original system while learning a comparable model. Further,

It can be **clearly observed** that the proposed database based systems that uses modes are significantly faster than the original BoostR system. However, this performance is achieved without significantly losing learning accuracy. Hence, **Q1** can be answered affirmatively in that the proposed methods are significantly faster than the state-of-the-art baseline. **Q2** can be answered negatively in that we do not sacrifice learning performance for improved learning time.

Conclusion and Future Work

We considered the problem of scaling up a successful boosting algorithm for SRL models. To this effect, we designed a in-memory database solution that exploited the search bias used in many logical models. Our initial evaluations clearly demonstrate that this learning system is capable of learning accurate models in significantly shorter amount of time.

Extensive evaluations of this approach is our next immediate direction for future research. Employing approximate counts for the groundings will potentially allow for even greater savings in time. However, these approximations need to be theoretically analysed for the learning performance, another interesting research direction. Finally, embedding the powerful structure learning approach such as boosting inside a large-scale system such as DeepDive will allow us to fully realize the gains attained in related research fields.

References

Bilenko, M., and Mooney, R. 2003. Adaptive duplicate detection using learnable string similarity measures. In *KDD*.

Craven, M.; DiPasquo, D.; Freitag, D.; McCallum, A.; Mitchell, T.; Nigam, K.; and Slattery, S. 1998. Learning to extract symbolic knowledge from the World Wide Web. In *AAAI*, 509–516.

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

Friedman, J. 2001. Greedy function approximation: A gradient boosting machine. *Annals of Statistics* 29.

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

Khot, T.; Natarajan, S.; Kersting, K.; and Shavlik, J. 2011. Learning markov logic networks via functional gradient boosting. In *ICDM*.

Kok, S., and Domingos, P. 2009. Learning Markov logic network structure via hypergraph lifting. In *ICML*.

Kok, S., and Domingos, P. 2010. Learning Markov logic networks using structural motifs. In *ICML*.

Lavrac, N., and Dzeroski, S. 1994. *Inductive logic programming - techniques and applications*. Ellis Horwood series in artificial intelligence. Ellis Horwood.

Mihalkova, L.; Huynh, T.; and Mooney, R. 2007. Mapping and revising markov logic networks for transfer learning. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 1*.

Milch, B.; Marthi, B.; and Russell, S. 2004. Blog: Relational modeling with unknown objects. In *Proceedings of the SRL Workshop in ICML*.

Natarajan, S.; Khot, T.; Kersting, K.; Gutmann, B.; and Shavlik, J. 2012. Gradient-based boosting for statistical relational learning: The Relational Dependency Network case. *MLJ*.

Neville, J., and Jensen, D. 2007. Relational dependency networks. *Introduction to Statistical Relational Learning*.

Niu, F.; Zhang, C.; Re, C.; and Shavlik, J. 2012. Scaling inference for markov logic via dual decomposition. In *ICDM*, 1032–1037.

Poon, H., and Domingos, P. 2007. Joint inference in information extraction. In *AAAI*, 913–918.

Poyrekar, S.; Natarajan, S.; and Kersting, K. 2014. A deeper empirical analysis of CBP algorithm: Grounding is the bottleneck. In *Statistical Relational Artificial Intelligence, Papers from the 2014 AAAI Workshop, Québec City, Québec, Canada, July 27, 2014*.

Schulte, O., and Qian, Z. 2015. SQL for SRL: structure learning inside a database system. *CoRR* abs/1507.00646.

Shavlik, J., and Natarajan, S. 2009. Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network. In *IJCAI*.

Zeng, Q.; Patel, J. M.; and Page, D. 2014. Quickfoil: Scalable inductive logic programming. *Proc. VLDB Endow.* 8(3).