Data Mining and Knowledge Discovery (2021) 35:2089–2111
https://doi.org/10.1007/s10618-021-00770-8

Structure learning for relational logistic regression: an ensemble approach

Nandini Ramanan[1] · Gautam Kunapuli[2] · Tushar Khot[3] · Bahare Fatemi[4] · Seyed Mehran Kazemi[4] · David Poole[4] · Kristian Kersting[5] · Sriraam Natarajan[2]

Received: 15 April 2019 / Accepted: 26 May 2021 / Published online: 14 July 2021

Abstract

We consider the problem of learning Relational Logistic Regression (RLR). Unlike standard logistic regression, the features of RLR are first-order formulae with associated weight vectors instead of scalar weights. We turn the problem of learning RLR to learning these vector-weighted formulae and develop a learning algorithm based on the recently successful functional-gradient boosting methods for probabilistic logic models. We derive the functional gradients and show how weights can be learned simultaneously in an efficient manner. Our empirical evaluation on standard data sets demonstrates the superiority of our approach over other methods for learning RLR.

Keywords Statistical relational learning · Boosting · Relational models · Functional Gradient Boosting · Probabilistic Machine Learning

1 Introduction

Statistical Relational Learning models (SRL) combine the representational power of logic with the ability of probability theory specifically, and statistical models in general to model noise and uncertainty (Getoor and Taskar 2007; Raedt et al. 2016). They have

Responsible editor: Hendrik Blockeel.

✉ Nandini Ramanan
nramanan@iu.edu

[1] Indiana University, Bloomington, IN, USA

[2] The University of Texas at Dallas, Richardson, TX, USA

[3] Allen Institute for Artificial Intelligence, Seattle, WA, USA

[4] University of British Columbia, Vancouver, BC, USA

[5] TU Dortmund University, Dortmund, Germany

generally ranged from directed models (Kersting and De Raedt 2007; Koller 1999; Heckerman et al. 2007; Getoor and Taskar 2007; Kazemi and Poole 2018) to undirected models (Richardson and Domingos 2006; Taskar et al. 2007; Kimmig et al. 2012). We consider the more recent, well-understood directed model of Relational Logistic Regression (RLR) (Kazemi et al. 2014a; Fatemi et al. 2016). One of the key attributes of RLR is that they are population size invariant (Poole et al. 2014), i.e., they scale well with increase in population sizes unlike other methods such as Markov Logic Networks (Domingos and Lowd 2009). Thus, they are a natural choice as a powerful modeling tool for many tasks (Popescul and Ungar 2003).

While the models are attractive from the modeling perspective, learning these models is computationally intensive. This is due to the fact that (like the field of Inductive Logic Programming) learning occurs at multiple levels of abstraction, that of the level of objects, sub-group of objects and relations and possibly at the individual instances of the objects. One could learn at the level of an individual, say ‘David‘ or at the level of Professors or at the level of Persons. Hence, most methods for learning these models have so far focused on the task of learning the so-called parameters (weights of the logistic function) where the rules (or relational features) are provided by the human expert and the data is merely used to learn the parameters.

Previous work on Structural Logistic Regression by Popescul and Ungar proposed a link prediction approach that generates and searches over a space of relational features using the Bayesian Information Criterion (BIC) score to learn the best set of link predictors (Popescul and Ungar 2003). However, the complexity of searching a large feature space and avoiding over-fitting makes it difficult to be employed on large data sets. In other supervised learning approaches, a “flat” feature vector representation for each link is created. Thus, the prediction made for the possible set of links is independent of the other predictions. Our approach explicitly does not make this assumption and allows for learning in the presence of rich relational data.

We consider the problem of full model learning, also known as structure learning of RLR models. A simple solution to learning these models could be to learn the rules separately using a logic learner and then employ the parameter learning strategies (Huynh and Mooney 2008). Huynh and Mooney use ALEPH (Srinivasan 2001) to learn the structure, followed by L1-regularized logistic regression to learn the weights for automatic feature selection. While reasonably easy to implement, the key issue is that the disconnect between rule and parameter learning can result in poor predictive performance as shown repeatedly in the literature (Natarajan et al. 2012; Richardson and Domingos 2006). Inspired by the recent successes of non-parametric learning methods for SRL models (Gutmann and Kersting 2006; Kersting and Driessens 2008; Natarajan et al. 2017, 2011, 2012), we develop a non-parametric learning method for full model learning (structure and parameters) of RLR models.

More specifically, we develop a gradient-boosting technique for learning RLR models. We derive the gradients for the different weights of RLR and show how the rules of the logistic function are learned simultaneously with their corresponding weights. Unlike the standard adaptations of the functional gradients, RLR requires learning a different set of weights per rule in each gradient step and hence requires learning multiple weights jointly for a single rule. As we explain later, the gradients correspond to a set of vector weighted clauses that are learned in a sequential manner.

Each clause can be seen as a relational feature for the logistic function. We also note that RLR can be viewed as a probabilistic combination function in that it can stochastically combine the distributions due to different set of parents (in graphical model terminology). Hence, if our learning technique is employed in the context of learning joint models, our work can be seen as a new interpretation of learning boosted Relational Dependency Networks (RDNs) (Getoor and Taskar 2007; Natarajan et al. 2012), where the standard aggregators are replaced with a logistic regression combination function which could potentially yield interesting insights into directed SRL models. We demonstrate the effectiveness of this combination function on real data sets and compare against several baselines including the state-of-the-art MLN learning algorithms.

The rest of the paper is organized as follows: first we introduce the necessary background and the notations. Next, we derive the gradients and present the algorithm for learning RLR models. Finally, we conclude the paper by presenting our extensive experimental evaluations on standard SRL data sets and outlining the areas for future research.

2 Background and notation

In this section, we define our notation and provide necessary background for our readers to follow the rest of the paper. Throughout the paper, we assume $True$ is represented by 1 and $False$ is represented by 0.

2.1 Logistic regression

Let Q be a Boolean random variable with range $\{1, 0\}$ whose value depends on a set $\{X_1, X_2, \ldots, X_n\}$ of random variables. Logistic regression (McCullagh 1984) defines the conditional probability of Q given $X_1, X_2, \ldots, X_n$ as the sigmoid of a weighted sum of X_is:

$$Prob(Q = 1 \mid X_1, \ldots, X_n) = \sigma(w_0 + w_1 X_1 + \cdots + w_n X_n) \tag{1}$$

where $\sigma(z) = 1/(1 + \exp(-z))$ is the sigmoid function.

2.2 Finite-domain, function-free, first-order logic

A population is a finite set of objects. We assume for every object, there is a unique constant denoting that object. A logical variable (logvar) is typed with a population. A term is a logvar or a constant. We show logvars with lower-case letters (e.g., x) , constants with upper-case letters (e.g., X), the population associated with a logvar x with Π_x, and the size/cardinality of the population with $|\Pi_x|$. A lower-case letter in bold represents a tuple of logvars (e.g., $\mathbf{x}$), and an upper-case letter in bold is a tuple of constants (e.g., $\mathbf{X}$).

An atom is of the form $\mathsf{Q}(t_1, \ldots, t_k)$ where Q is a functor and each t_i is a term. When $range(\mathsf{Q}) = \{1, 0\}$, Q is a predicate. A substitution is of the form $\theta = \{\langle x_1, \ldots, x_k\rangle / \langle t_1, \ldots, t_k\rangle\}$ where x_is are logvars and t_is are terms. A grounding of an atom with logvars $x_1, \ldots, x_k$ is a substitution $\{\langle x_1, \ldots, x_k\rangle / \langle X_1, \ldots, X_k\rangle\}$ mapping each of its logvars to a constant in the population of the logvar. For a set $\mathcal{A}$ of atoms, $\mathcal{G}(\mathcal{A})$ represents the set of all possible groundings for the atoms in $\mathcal{A}$. A literal is an atom or its negation. A formula φ is a literal, the conjunction of two formulae $\varphi_1 \wedge \varphi_2$, or a disjunction of two formulae $\varphi_1 \vee \varphi_2$. The application of a substitution $\theta = \{\langle x_1, \ldots, x_k\rangle / \langle t_1, \ldots, t_k\rangle\}$ on a formula φ is represented as $\varphi\theta$ and replaces each x_i in φ with t_i. An instance of a formula φ is obtained by replacing each logvar x in φ by one of the objects in Π_x. A conjunctive formula contains no disjunction. A weighted formula (WF) is a triple $\langle \varphi, w_T, w_F\rangle$ where φ is a formula and w_T and w_F are real numbers.

2.3 Relational logistic regression

Let $\mathsf{Q}(\mathbf{x})$ be a Boolean atom whose probability depends on a set $\mathcal{A}$ of atoms such that $\mathsf{Q} \notin \mathcal{A}$ i.e. we do not allow recursive clauses. We refer to $\mathcal{A}$ as the parents of Q. Let ψ be a set of WFs containing only atoms from $\mathcal{A}$, J be a function from groundings in $\mathcal{G}(\mathcal{A})$ to truth values, and $\theta = \{\mathbf{x}/\mathbf{X}\}$ be a substitution from logvars in $\mathbf{x}$ to constants in $\mathbf{X}$. Relational logistic regression (RLR) (Kazemi et al. 2014a) defines the probability of $\mathsf{Q}(\mathbf{X})$ given J as follows:

$$
\begin{aligned}
&Prob_{\psi}(\mathsf{Q}(\mathbf{X}) = 1 \mid J) \\
&\quad = \sigma\left(w_0 + \sum_{\langle \varphi, w_T, w_F\rangle \in \psi} w_T \cdot \eta_T(\varphi\theta, J) + w_F \cdot \eta_F(\varphi\theta, J)\right)
\end{aligned} \tag{2}
$$

where w_0 is a bias/intercept, $\eta_T(\varphi\theta, J)$ is the number of instances of $\varphi\theta$ that are true with respect to J, and $\eta_F(\varphi\theta, J)$ is the number of instances of $\varphi\theta$ that are false with respect to J. Note that $\eta_T(\mathsf{True}, J) = 1$. Also note that the bias can be considered as a WF whose formula is $True$. Following the work by Kazemi et al. (2014a), without loss of generality we assume the formulae of all WFs for RLR models are conjunctive.

Example 1 Let $\mathsf{active}(p)$, $\mathsf{advisedBy}(s, p)$, and $\mathsf{phd}(s)$ be three atoms representing respectively whether a professor is active, whether a student is advised by a professor, and whether a student is a PhD student. Suppose we want to define the conditional probability of $\mathsf{active}(p)$ given the atoms $\mathcal{A} = \{\mathsf{advisedBy}(s, p), \mathsf{phd}(s)\}$. Consider an RLR model with an intercept of -3.5 which uses only the following WF to define this conditional probability:

$$\psi = \{\langle \mathsf{advisedBy}(s, p) \wedge \mathsf{phd}(s), 1, 0\rangle\}$$

According to this model, for an assignment J of truth values to $\mathcal{G}(\mathcal{A})$:

$$\begin{aligned} &Prob_{\psi}(\mathsf{active}(P) = 1 \mid J) \\ &\quad = \sigma(-3.5 + 1 \cdot \eta_T(\mathsf{advisedBy}(s, P) \wedge \mathsf{phd}(s), J)), \end{aligned}$$

where $\eta_T(\mathsf{advisedBy}(s, P) \wedge \mathsf{phd}(s), J)) = \#S \in \Pi_s$ s.t. $\mathsf{advisedBy}(S, P) \wedge \mathsf{phd}(S)$ according to J, corresponding to the number of PhD students advised by P. We denote the count for instances of satisfied groundings as #. When this count is greater than or equal to 4, the probability of P being an active professor is closer to one than zero; otherwise, the probability is closer to zero than one. Therefore, this RLR model represents "a professor is active if the professor advises at least 4 PhD students".

With this background on Relational Logistic Regression, we introduce the Functional Gradient Boosting paradigm in the following section. This enables us to formulate a learning problem for RLR in which we learn both the structure and the parameters simultaneously.

2.4 Functional gradient boosting

We discuss the functional gradient boosting (FGB) approach in the context of relational models. This approach is motivated by the intuition that finding many rough rules-of-thumb of how to change one's probabilistic predictions locally can be much easier than finding a single, highly accurate model. Specifically, this approach turns the problem of learning relational models into a series of relational function approximation problems using the ensemble method of gradient-based boosting. This is achieved by the application of the work by Friedman (2001) on gradient boosting to SRL models. That is, we represent the conditional probability distribution as a weighted sum of regression models that are grown via a stage-wise optimization (Natarajan et al. 2012; Khot et al. 2011).

The conditional probability of an example y_i[1] depends on its parents $\mathbf{x}_i = \mathbf{parents}(y_i)$. The goal of learning is to fit a model $Prob(y \mid \mathbf{x}) \propto e^{\psi(y;\mathbf{x})}$, which can be expressed non-parametrically in terms of a potential function $\psi(y_i;\, \mathbf{x}_i)$:

$$Prob(y_i \,|\mathbf{x}_i) = \frac{e^{\psi(y_i;\, \mathbf{x}_i)}}{\sum_{y'} e^{\psi(y';\, \mathbf{x}_i)}} \tag{3}$$

At a high-level, we are interested in successively approximating the function ψ as a sum of weak learners, which are relational regression clauses, in our setting. Functional gradient ascent starts with an initial potential ψ_0 and iteratively adds gradients Δ_i. After m iterations, the potential is given by $\psi_m = \psi_0 + \Delta_1 + \cdots + \Delta_m$. Here, Δ_m is the functional gradient at episode m and is

$$\Delta_m = \nu_m \cdot E_{\mathbf{x},y}\left[\frac{\partial}{\partial \psi_{m-1}} \log\, Prob(y \mid \mathbf{x};\ \psi_{m-1})\right], \tag{4}$$

[1] We use the term example to mean the grounded target literal. Hence $y_i = 1$ denotes that the grounding $\mathsf{Q}(\mathbf{X}) = 1$ i.e., the grounded target predicate is true. Following standard Bayesian networks terminology, we denote the parents $\mathcal{A}(\mathsf{Q})$ to include the set of formulae ψ that influence the current predicate Q.

where ν_m is the learning rate. Given that the expectation $E_{x,y}[..]$ cannot be computed exactly as the joint distribution $Prob(\mathbf{x}, \mathbf{y})$ is unknown, functional gradient based methods treat the data as a proxy for this joint distribution (Dietterich et al. 2004). More precisely, Dietterich et al. suggested evaluating the gradient for each training example and fitting a small regression function (potentially a tree) to the pseudo-residuals from a regression at each iteration i.e., fit a regression tree h_m on the training examples $[(x_i, y_i), \Delta_m(y_i; x_i)]$. They 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). Thus, ascent in the direction of h_m will approximate the true functional gradient.

Note that in the functional gradient presented in Eq. (4), the expectation $E_{\mathbf{x},y}$ cannot be computed as the joint distribution $Prob(\mathbf{x},\ y)$ is unknown. Instead of computing the functional gradients over the potential function, they are instead computed pointwise for each labeled training example i: $\langle \mathbf{x}_i,\ y_i \rangle$. Now, this set of local gradients become the training examples to learn a weak regression model that approximates the gradient Δ_m at stage m.

The functional gradient with respect to $\psi(y_i = 1;\ \mathbf{x}_i)$ of the likelihood for each example $\langle y_i, \mathbf{x}_i \rangle$ can be shown to be:

$$\frac{\partial \log Prob(y_i;\ \mathbf{x}_i)}{\partial \psi(y_i = 1;\ \mathbf{x}_i)} = \mathbb{I}(y_i = 1; \mathbf{x}_i) - Prob(y_i = 1; \mathbf{x}_i) \tag{5}$$

where $\mathbb{I}$ is the indicator function, that is 1, if $y_i = 1$, and 0 otherwise. This expression is simply *the adjustment required to match the predicted probability with the true label of the example*. If the example is positive and the predicted probability is less than 1, this gradient is positive indicating that the predicted probability should move towards 1. Conversely, if the example is negative and the predicted probability is greater than 0, the gradient is negative driving the value the other way.

This elegant gradient expression might appear simple, but in fact, naturally and intuitively captures, example-wise, the general direction that the overall model should be grown in. The $I - P$ form of the functional gradients is a consequence of the sigmoid function as a modeling choice, and is a defining characteristic of FGB methods. As we show below, our proposed approach to RLR also has a similar form. The significant difference, however, is in the novel definition of the potential function ψ.

In prior work, relational regression trees (RRTs) (Blockeel 1999) were used to fit the gradient function Δ_m to the pointwise gradients for every training example. Each RRT can be viewed as defining several new feature combinations, one corresponding to each path from the root to a leaf. A key difference in our work is that we employ the use of weighted formulae (vector-weighted clauses,[2] to be precise) as we explain later. From this perspective, our work is closer to the boosting MLN work that employed the use of weighted clauses (Khot et al. 2011). We generalize this by learning a weight vector per clause that allows for a more compact representation of the true and false instances of the formula. An example of a weighted clause is provided in Fig. 1 where there are four clauses for predicting *advisedBy(A,B)*.

[2] We use formulae and clauses interchangeably from hereon.

$$
\begin{aligned}
&w_1 : professor(b) \wedge professor(a) \rightarrow advisedBy(a, b)\\
&w_2 : professor(b) \wedge \neg professor(a) \wedge coauthor(a, b) \rightarrow advisedBy(a, b)\\
&w_3 : professor(b) \wedge \neg professor(a) \wedge \neg coauthor(a, b) \rightarrow advisedBy(a, b)\\
&w_4 : \neg professor(b) \rightarrow advisedBy(a, b)
\end{aligned}
$$

Fig. 1 Example of relational regression clauses (RRCs). The task was to predict if A is AdvisedBy B, given the relations of people at a university

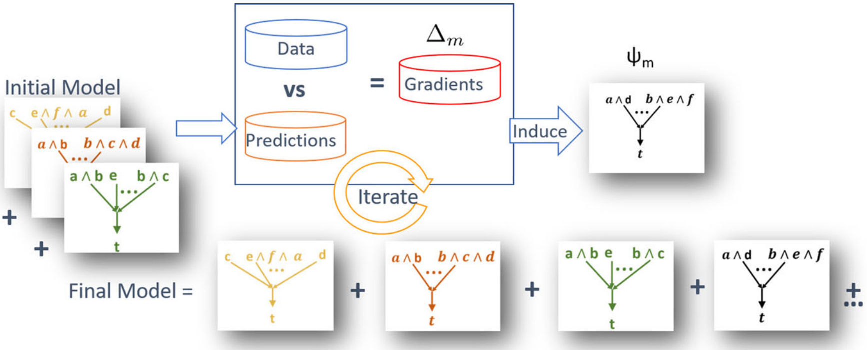

Fig. 2 Relational Functional Gradient Boosting. This is similar to standard functional gradient boosting (FGB) where trees are induced stage-wise; the key difference is that these trees are RRCs

Note that while we show that the standard weighted clauses are similar to a MLN, our weighted clauses have *an important distinction*. Corresponding to each clause is a weight vector instead of a single scalar weight which captures the weight of true groundings, false groundings and the uninformed prior weights of the clause. The gradient-boosting that we develop in the next section builds upon these clauses and as mentioned earlier is similar to MLN boosting with the key difference being that instead of learning one weight per clause, we learn three weights in the vector.

The key intuition with boosting regression clauses is that each clause will define a new feature combination and the different clauses together capture the *latent relationships* that are learned from the data. While the final model itself is linear (as it is the sum of the weighted groundings of all the clauses), the clauses themselves define richer features thus allowing for learning a more complex model than a simple linear one. Figure 2 presents the schematic for boosting. The idea is that first a regression function (shown as a set of clauses) is learned from the training examples and these clauses are then used to determine the gradients (weights) of each example in the next iteration. The gradient is typically computed in the prior work as $I - P$. Once the examples are weighted, a new set of clauses are induced from them. These clauses are then considered together and the *regression values are added* when weighing the examples and the process is iterated.

There are several benefits of the boosting approach for learning RLR models. First, being a non-parametric approach (i.e., the model size is not chosen in advance), the number of parameters naturally grows as the number of training episodes increases. In turn, relational features as clauses are introduced only as necessary, so that a potentially large search space is not explicitly considered. Second, such an algorithm is fast and straightforward to implement. One could potentially employ any relational regression learner in the inner loop to learn several types of models. Third, as with previous relational models, the use of boosting for learning RLR models makes it possible to learn the structure and parameters simultaneously making them an attractive choice for learning from large scale data sets (Malec et al. 2016; Yang et al. 2017).

3 Functional gradient boosting for RLR

3.1 Preliminaries

Given the background on RLR and the gradient-boosting approach, we now focus on the learning task of RLR. Let us rewrite the conditional probability of an example y given weighted formulae $\langle \varphi_1, w_{T1}, w_{F1} \rangle, \cdots, \langle \varphi_k, w_{Tk}, w_{Fk} \rangle$ corresponding parents $R_1, \cdots, R_k$ in the RLR model as:

$$\begin{aligned} Prob(\mathbf{y} = 1 \mid \mathsf{R}_1, \cdots, \mathsf{R}_k) = \sigma(w_0 & \\ & + w_{T1}\eta_T(\varphi_1\theta, R_1) + w_{F1} \cdot \eta_F(\varphi_1\theta, R_1) \\ & + \cdots \\ & + w_{Tk}\eta_T(\varphi_k\theta, R_k) + w_{Fk} \cdot \eta_F(\varphi_k\theta, R_k)) \end{aligned} \tag{6}$$

where $\sigma(\cdot)$ is the sigmoid function. For example, let $y = \mathsf{Popularity}(a)$ indicate the popularity of a professor a. Consider two formulae ϕ_1=$\mathsf{Publication}(A, P)$ and ϕ_2=AdvisedBy(A, S). The weights of the first formula control the influence of the number of publications on the popularity of the professor where $R_1 = \mathsf{Publication}(a, P)$. Similarly the second formula controls the influence of the number of students advised by the professor. For learning a model for RLR, we thus need to learn these clauses ϕ_i and their weights w_{Ti}, w_{Fi} (the parents are determined by the structure of the clause). Also, we can assume that the bias term w_0 can be part of the weight vectors for all the learned clauses. This allows a greedy approach that incrementally adds new clauses, such as FGB, to automatically update the bias term by learning w_0 for each new clause. Our learning problem can be defined as:

Given: A set of grounded facts and the corresponding positive and negative grounded literals (examples)

To Do: Learn the best set of formulae φ_i with their corresponding weight vector $\mathbf{w}_i = [w_0, w_1, w_2]$ that minimizes the loss function.

To simplify the learning problem, we introduce vector-weighted clauses (formulae), denoted as $\mathbf{w}$: Clause, that are a generalization of traditional weighted clauses with single weights. More specifically, our weighted clauses have three dimensions, i.e., $\mathbf{w} = [w_0,\ w_1,\ w_2]^T$, where w_0 is a bias/intercept, w_1 is the weight over the satisfiable groundings of the current clause (analogous to w_{Ti}) and w_2 is the weight of the unsatisfiable groundings of the current clause (analogous to w_{Fi}). We also use a short hand notation t_i and f_i for the two grounding counts $\eta_T(\varphi_i\theta, R_1)$ and $\eta_F(\varphi_i\theta, R_1)$ respectively in Eq. (6).

Example 2 Consider an RLR model for defining the conditional probability of $y = \mathsf{popularity}(a)$ which has only one WF:

$$\begin{aligned}&\langle \mathsf{publication}(a, b), w_T, w_F\rangle\\ &\quad\equiv [w_0, w_1, w_2] : \mathsf{popularity}(a){:}{-}\mathsf{publication}(a, b)\end{aligned}$$

Let $t_y = \sum_b \mathsf{publication}(a,\ b)$ be the number of instances of b for which $\mathsf{publication}(a,\ b)$ is true for the current grounding of y, and let $f_y = \sum_b (1 - \mathsf{publication}(a,\ b))$ be the number of instances of b for which $\mathsf{publication}(a,\ b)$ is false for the example y. Using vector-weighted clauses in the RLR model, we can compute

$$\begin{aligned}&Prob(\mathsf{popularity}(a)|\mathsf{publication}(a,\ B))\\ &\quad= \sigma\left(w_0 + w_1 \cdot t_y + w_2 \cdot f_y\right), \forall a.\end{aligned}$$

3.2 RFGB for RLR

Our goal is to learn the full structure of the model, which involves learning two concepts – the structure (formulae/clauses) and their associate parameters (the weight vectors). To adapt functional gradient boosting to the task of learning RLR, we map this probability definition over the parameter space w_0, w_1, w_2 to the functional space, ψ:

$$\begin{aligned}Prob(y_i = 1|\mathbf{x}_i) &= \sigma(\psi\,(y_i; \mathbf{x}_i))\\ &= \sigma\,(w_0 + w_1 \cdot t_i + w_2 \cdot f_i)\end{aligned} \tag{7}$$

Recall that in FGB, the gradients of the likelihood function with respect to the potential function are computed separately for each example. Correspondingly, the regression function ψ for the i-th example needs to be clearly defined and is:

$$\psi(y_i;\ \mathbf{x}_i) = w_0 + w_1 \cdot t_i + w_2 \cdot f_i. \tag{8}$$

The key difference to the existing gradient boosting methods for RDNs (Natarajan et al. 2012) and MLNs (Khot et al. 2011) is that the RLR learning algorithm needs to learn a weight vector per clause instead of a single weight.

Also, recall that while in traditional parametric gradient-descent, one would compute the parametric gradient over the loss function and iteratively update the parameters with these gradients, for gradient boosting, we first compute the functional gradients over the log-likelihood given by:

$$\Delta(y_i;\ \mathbf{x}_i) = \mathbb{I}(y_i = 1) - Prob(y_i = 1;\ \mathbf{x}_i) \tag{9}$$

where $\mathbb{I}$ is the indicator function. As with other relational functional gradients (see Sect. 2.4), this elegant expression naturally falls out when the log-likelihood of the sigmoid is differentiated with respect to the function. As before, the gradient is simply the adjustment required for the probabilities to match the observed value (y_i) in the training data for each example. Note that this is simply the outer gradient, that is, the gradient is computed for each example and a single vector-weighted clause needs to be learned for this set of gradients. While learning the clause itself, we must optimize a different loss function as we show next.

In order to generalize beyond the training examples, we fit a regression function ψ (which is essentially a *vector-weighted clause*) over the training examples such that the squared error between $\psi(y_i;\, \mathbf{x}_i)$ and the functional gradient $\Delta(y_i;\, \mathbf{x}_i)$ is minimized over all the examples. The inner loop thus amounts to learning vector-weighted clauses such that we minimize the (regularized) squared error between the RLR model and the functional gradients over the n training examples:

$$\begin{aligned}\underset{w_0, w_1, w_2}{\text{minimize}} \sum_{i=1}^{n} (w_0 + w_1 t_i + w_2 f_i - \Delta(y_i; \mathbf{x}_i))^2 \\ + \lambda(w_0^2 + w_1^2 + w_2^2),\end{aligned} \tag{10}$$

where $\lambda > 0$ is a regularization parameter. In principle, λ can be chosen using a line search with a validation set when the size of the data sets are large. However, in our data sets, we only consider our λ from within the set $\langle 10^2, \ldots, 10^{3.5}\rangle$. We pick the best λ using the tuning set and we report the performance on test. We also perform experiments to demonstrate that as long as λ is within the set $\langle 10^2, \ldots, 10^{3.5}\rangle$, our algorithm is not sensitive in most domains.

Close inspection of the loss function above reveals that solving this optimization problem amounts to fitting count features: $\mathbf{c}_i = [1,\ t_i,\ f_i]$ for each grounded example i to the corresponding functional gradient, Δ_i. Note that the Eq. (10) can be viewed as a regularized least-squares regression problem to identify weights $\mathbf{w} = [w_0,\ w_1,\ w_2]^T$. The problem (10) can be written in vector form as

$$\min_{\mathbf{w}} \ \|C\mathbf{w} - \Delta\|^2 + \lambda\|\mathbf{w}\|^2$$

where the i-th row of the count matrix C are the count features $\mathbf{c}_i$ of the i-th example. This problem has a closed-form solution that can be computed efficiently:

$$\mathbf{w} = (C^T C + \lambda I)^{-1} C^T \Delta. \tag{11}$$

The quantity $C^T C$ captures the count covariance across examples, while the quantity $C^T \Delta$ captures the count-weighted gradients:

$$\begin{aligned} C^T C &= \begin{bmatrix} n & \sum_{i=1}^n t_i & \sum_{i=1}^n f_i \\ \sum_{i=1}^n t_i & \sum_{i=1}^n t_i^2 & \sum_{i=1}^n t_i f_i \\ \sum_{i=1}^n f_i & \sum_{i=1}^n t_i f_i & \sum_{i=1}^n f_i^2 \end{bmatrix}, \\ C^T \Delta &= \begin{bmatrix} \sum_{i=1}^n \Delta_i \\ \sum_{i=1}^n t_i \Delta_i \\ \sum_{i=1}^n f_i \Delta_i \end{bmatrix}. \end{aligned} \tag{12}$$

In this manner, functional gradient boosting enables a natural combination of conditionals over all the examples. This weight update forms the backbone of our approach: boosted relational logistic regression or B- RLR.

3.3 Algorithm for B-RLR

Algorithm 1 Boosted Relational Logistic Regression (B- RLR) learning

```
1: function B- RLR($Data$,$p$)
2:     where $Data = \{\langle X, Y\rangle\}$
3:     $F_0 := \gamma$
4:     for $1 \leq m \leq M$ do                                   ▷ M gradient steps
5:         $F_m := F_{m-1}$
6:         for $y \in \mathbf{Y_p}$ do                             ▷ Iterate through target predicates
7:             $S_p :=$ COMPUTEGRADIENTS($y$, $Data$, $F_m$)       ▷ Compute gradients, $\Delta_i$ for $y_i$
8:             $\hat{\psi}_m :=$ FITREGRESSION($S_p$, $Data$, $y$) ▷ Learn vector-weighted regression clause
9:             $F_m := F_m + \hat{\psi}_m$                         ▷ Update model
10:        end for
11:    end for
12:    return $F_m$
13: end function
```

We outline the algorithm for boosted RLR (B- RLR) learning in Algorithm 1. We initialize the regression function with an uniform prior γ i.e. $F_0(y_i) = \gamma$ (line 2). Given the input training examples Y which correspond to the grounded instances of the target predicate y and the set of facts, i,e., the grounded set of all other predicates (denoted as X) in the domain, the goal is to learn the set of vector-weighted clauses that influence the target predicate.

Since there could potentially be multiple target predicates (when learning a joint model such as a RDN where each influence relation is an RLR), we denote the current predicate as y from $\mathbf{Y_p}$. In the m^{th} iteration of functional-gradient boosting, we compute the functional gradients for these examples using the current model F_m and the parents of $\mathbf{Y_p}$ as per this model (line 7). Given these regression examples S_p, we learn a vector-weighted clause using FITREGRESSION. Our algorithm is restricted to learn only non-recursive Horn clauses. FITREGRESSION uses all the other facts X to learn the structure and parameters of the clause. We then add this regression function, $\hat{\psi}_m$ approximating the functional gradients to the current model, F_m. We repeat this over M iterations where M is typically set to 10 in our experiments.

Next, we describe FITREGRESSION to learn vector-weighted clauses from input regression examples S, Data D and target predicate $\mathsf{p}(x)$ in Algorithm 2. We initialize the vector-weighted clause with empty body and zero weights i.e. $[0, 0, 0]^T : y{:}-\varnothing$. We first create all possible literals that can be added to the clause given the current body (line 5). We use modes (Muggleton and De Raedt 1994) from inductive logic programming (ILP) to efficiently find the relevant literals here.

For each literal l in this set, we calculate the true and false groundings for the newly generated clause by adding the literal to the body (line 9). To perform this calculation, we ground the left hand side of the horn clause (i.e., the query literal) and count the number of groundings of the body corresponding to the query grounding. For instance if the grounding of the query is *advisedBy(John,Mary)* corresponding to *advisedBy(student,prof)*, then we count the number of instances of the body that correspond to *John* and *Mary*. If the body contains the publications in common, they are counted accordingly. If the body is about courses John took, they are counted correspondingly. This is similar to counting in any relational probabilistic model such as MLNs or BLPs (Kok and Domingos 2005; Kersting and De Raedt 2007). Following standard SRL models, we assume closed-world. This allows us to deduce the number of false groundings as the difference between the total number of possible groundings and the number of counted (positive) groundings.

We can then calculate the count matrix C and weights $\mathbf{w}$ as described earlier (line 11–12). We score each literal based on the squared error and greedily pick the best scoring literal $\hat{l}$. We repeat this process till the clause reaches its maximum allowed length (set to 4 in our experiments).

Algorithm 2 Vector-weighted regression clause learning

```
1: function FitRegression($S$, $D$, p)
2:     where $S = \{\langle y, \Delta \rangle\}$
3:     where $D = \{\langle X, Y \rangle\}$
4:     body :=$\emptyset$; $\mathbf{w}$:=$[0, 0, 0]^T$                    ▷ Initialize empty clause
5:     while $len$(body) $\leq$ $clause\_length$ do
6:         $L$ := PossibleLiterals(p(x), body)  ▷ Initialize with target predicate p(x) & Generate potential literals
7:         for $\ell \in L$ do                                          ▷ Score each literal
8:             clause := ‘y :− body $\wedge$ $\ell$.’
9:             $C$:=0
10:            for $x_i \in X$ do                          ▷ Calculate groundings per example
11:                $t_i, f_i$ = CalculateGroundings($y$, $x_i$, clause)
12:            end for
13:            $C$ := CreateCountMatrix($\{t_i, f_i\}$)
14:            $\mathbf{w}(\ell) := (C^T C + \lambda I)^{-1} C^T \Delta$.     ▷ $\Delta$ is gradient & $\lambda$ is user-defined parameter
15:            score($\ell$) := ScoreFit($\mathbf{w}(\ell)$, $\Delta$)
16:        end for
17:        $\hat{\ell} := \arg\min_{\ell}$ score($\ell$)
18:        $\mathbf{w} := \mathbf{w}(\hat{\ell})$
19:        body := body $\wedge$ $\hat{\ell}$
20:    end while
21:    return $\mathbf{w}$: y :− body.
22: end function
```

To summarize, given a target, the algorithm computes the gradient for all the examples based on the expression $I - P$. Given these gradients, the inner loop searches over the set of weak predictors (we restrict the possible clauses to have a maximum length of 4) such that the MSE is minimized. The resulting vector-weighted clauses are then added to the set of formula and are then used for the subsequent steps of gradient computations. The procedure is repeated until a preset number of formulae are learned. The search for the most optimal clause can be guided by experts by providing relevant search information as modes (Muggleton and De Raedt 1994). The overall procedure is similar to learning RDNs (Natarajan et al. 2012) and MLNs (Khot et al.

2011) with two significant differences - the need for multiple weights in the clauses and correspondingly the different optimization function inside the inner loop of the algorithm.

Given that we have outlined the B- RLR algorithm in detail, we now turn our focus to empirical evaluation of this algorithm.

4 Experiments and results

Our experiments will aim to answer the following questions in order to demonstrate the benefits of B- RLR:

Q1 Does boosting RLR perform as well as or better than other relational methods?
Q2 Does the boosted method perform better than a significant feature engineered logistic regression approach?
Q3 Does functional gradient boosting perform better than the traditional learning approaches for clauses and weights?
Q4 Is the proposed approach sensitive with respect to the regularization constant, λ?

4.1 Methods considered

We now compare our $b - rlr$ approach to: (1) MLNs, we used the structure learning in Alchemy(http://alchemy.cs.washington.edu) (Kok and Domingos 2005) with the discriminative weight learning preconditioner scaled conjugate gradient (PSCG) Lowd and Domingos (2007). (2) the $agg - lr$ approach, which is propositional logistic regression model using the aggregate (relational) feature, (3) the $ilp - rlr$ approach where rules are learned using a logic learner (aleph), followed by weight learning for the formulae Fatemi et al. (2016), (4) $Aleph + +ExactL1$ approach is an improved discriminative methods for learning MLN clauses. It uses exact inference and L1-regularization instead of the normal L2 in order to encourage assigning zero weights to clauses returned by ALEPH++ (Huynh and Mooney 2008). The advantage of $Aleph + +ExactL1$ over $ilp - rlr$ comes from learning a large set of potential clauses, and (5) $mln - b$ which is a state-of-the-art boosted MLN structure learning method (Khot et al. 2011). We evaluate our approach on 1 synthetic data set and 4 real world data sets. Table 1 shows the sample aggregate (Relational) features constructed with the highest weights as generated by $agg - lr$.

In contrast to our approach, which performs parameter and structure learning *simultaneously*, the ILP- RLR baseline performs these steps *sequentially*. More specifically, we use PROGOL Muggleton (1995, 1996) for structure learning, followed by relational logistic regression for parameter learning. Table 2 shows the number of rules that were learned for each data set by PROGOL. Table 2 also shows some sample rules with the highest coverage scores as generated by PROGOL.

A natural question to ask is why not compare our method against the recently successful Boosted Relational Dependency Networks (Natarajan et al. 2012) (bRDN) method. We do not consider this comparison for two important reasons - first is that the $mln - b$ has already been compared against bRDN in the original work and

Table 1 This table shows the number of rules used by AGG-RLR for each data set as well as the features with high weights as picked by LR

Domains	Sample features
WorkedUnder (IMDB) 5 features constructed	count_genres_acted, count_movies_acted
AdvisedBy (UWCS) 8 features constructed	count_publications, count_taughtby
Female (Movie lens) 8 features constructed	count_movies, average_ratings
Cancer (SmCaFr) 3 features constructed	no_of_friends, no_of_friends_smoke
Faculty (WebKB) 4 features constructed	count_project, count_courseta

the conclusion was that they were nearly on par in performance in all the domains while bRDN is more efficient due to the use of existentials instead of counts when grounding clauses. Consequently, the second reason is that since our $agg-lr$ approach heavily employs counts, we considered the best learning method that employs counts as an aggregator, namely the $mln - b$ method. Our goal is not to demonstrate that $agg - lr$ is more effective than the well-known $mln - b$ or the bRDN approaches, but to demonstrate that boosting RLR does not sacrifice performance of learners and that RLR can be boosted as effectively as other relational probabilistic models.

4.2 Experimental setting

To keep comparisons as fair as possible, we used the following protocol: while employing $mln - b$ and $b - rlr$, we set the maximum number of clauses to 3, the beam-width to 10 and maximum clause length to 4. Similar configurations were adopted in our clause-learning setting. Gradient steps for $mln - b$ and $b - rlr$ were picked as per the performance. For all our baselines we used the standard settings as provided in the original work with minor modification for Alchemy alone. We set the evaluation function to **auto_m** with min score of an acceptable clause as 0.6. This configuration improved the overall accuracy in general.

4.3 Data sets

Smokes-cancer-friends: This is a synthetic data set, where the goal is to predict who has cancer based on the friends network of individuals and their observed smoking habits. The data set has three predicates: Friends, Smokes and Cancer. We evaluated the method over the Cancer predicate using the other predicates with 4-fold cross-validation and $\lambda = 10^{3.5}$.
UW-CSE: The UW-CSE data set (Richardson and Domingos 2006) was created from the University of Washington's Computer Science & Engineering department's student database and consists of details about professors, students and courses from 5 different subareas of computer science (AI, programming languages, theory, system and graphics). The data set includes predicates such as Professor, Student, Publication, AdvisedBy, HasPosition, TaughtBy etc., Our task is to learn, using the other predicates, to predict the AdvisedBy relation between a student and a professor. There are

Table 2 This table shows the number of rules used by ILP- RLR for each data set as well as the rules with the highest $\|P - N\|$ coverage as returned by PROGOL

Target (data set)	Sample rules generated for ILP- RLR using PROGOL
WorkedUnder (IMDB)	WorkedUnder(A, B) ⇐ isa(B, director), isa(A, actor),
6 *rules generated*	movie(C, A), movie(C, B).
	WorkedUnder(A, B) ⇐ genre(B, C), gender(A, male).
AdvisedBy (UWCS)	AdvisedBy(A, B) ⇐ hasPosition(B, E), inPhase(A, D),
16 *rules generated*	publication(C, A), publication(C, B).
	AdvisedBy(A, B) ⇐ hasPosition(B, D), inPhase(A, E),
	publication(F, A).
Female (Movie Lens)	Female(A) ⇐ tmpRatingArg1(B, A), tmpRatingArg2(B, C),
7 *rules generated*	genre(C, g4).
	Female(A) ⇐ age(A, 4), occupation(A, o14).
Cancer (SmCaFr)	Cancer(a) ⇐ friends(b, a), friends(b, c), smokes(c).
3 *rules generated*	Cancer(a) ⇐ smokes(a).
Faculty (WebKB)	Faculty(A) ⇐ courseProf(B, A), courseTA(B, C).
6 *rules generated*	Faculty(A) ⇐ courseProf(B, A), project(C, A),
	samePerson(A, A).

4, 106, 841 possible AdvisedBy relations out of which only 3380 are true. We employ 5-fold cross validation where we learn from four areas and predict on the other area with $\lambda = 10^{3.5}$ in our reported results.

IMDB: The IMDB data set was first used by Mihalkova and Mooney (2007) and contains five predicates: Actor, Director, Movie, Genre, Gender and WorkedUnder. We predict the WorkedUnder relation between an actor and director using the other predicates. Following Kok and Domingos (2009), we omitted the four equality predicates. We set $\lambda = 10^3$ and employed 5-fold cross-validation using the folds generation strategy suggested by Mihalkova and Mooney in Mihalkova and Mooney (2007) and averaged the results.

WebKB: The WebKB data set was first created by Craven et al. (1998) and contains information about department webpages and the links between them. It also contains the categories for each web-page and the words within each page. This data set was converted by Mihalkova and Mooney (2007) to contain only the category of each web-page and links between these pages. They created the following predicates: Student, Faculty, CourseTA, CourseProf, Project and SamePerson from these web-pages. We evaluated the method over the Faculty predicate using the other predicates and we performed 4-fold cross-validation where each fold corresponds to one university with set λ set as 10^2.

Movie lens: This is the well-known Movielens data set (Harper and Konstan 2015) containing information of about users, movies, the movies rated by each user containing user-movie pairs, and the actual rating the user has given to a movie. It contains predicates: Age, Genre, Occupation, Year, Ratings and Gender. In our experiments, we ignored the actual ratings and only considered whether a

Table 3 Details of relational domains used in our experiments

Data sets	Target	Types	Predicates	neg:pos Ratio
Sm-Ca-Fr	Cancer	1	3	1.32
IMDB	WorkedUnder	3	6	13.426
UW-CSE	AdvisedBy	9	12	539.629
WebKB	Faculty	3	6	4.159
Movie lens	FemaleGender	7	6	2.702

These data sets have high ratios of negative to positive examples, which is a key characteristic of relational data sets

movie was rated by a user or not. Also, since Gender can take only two values, we convert the Gender$(person, gender)$ predicate to a single arity predicate FemaleGender$(person)$. We learned B- RLR models for predicting FemaleGender using 5-fold cross-validation with $\lambda = 10^3$.

A key property of these relational data sets is the large number of negative examples. This is depicted in Table 3, which shows the size of various data sets used in our experiments. This is because, in relational settings, a vast majority of relations between objects are not true, and the number of negative examples far outnumbers the number of positive examples. In these data sets, simply measuring accuracy or log-likelihood can be misleading. Hence, we use metrics which are reliable in imbalanced setting like ours.

4.4 Results

We present the results of our experiments in Figs. 3 and 4, which compare the various methods on two metrics: area under the ROC curve (AUC-ROC) and area under the Precision-Recall curve (AUC-PR) respectively. From these figures, certain observations can be made clearly.

First, the proposed B- RLR method is on par or better than most methods across all data sets. Our approach achieves both significantly higher AUC-ROC as well AUC-PR than $Alchemy$ on all the domains. We are order of magnitude faster when compare to the MLN structure and weight learning in $Alchemy$. For both Movielens and Smokes-Cancer-Friends data sets, learning using Alchemy did not complete after a week. Hence, we did not report results on Alchemy for these two domains. On deeper inspection, it can be seen that the state-of-the-art boosting method for MLNs is more mixed in ROC-space when compared to B- RLR . However, it can be clearly seen that B- RLR performs better in PR-space.

Second, in the WebKB, MovieLens and Smokes-Cancer-Friends domain where we learn about a unary predicate, the performance of B- RLR is significantly better. This yields an interesting insight: RLR models can be natural aggregators over the associated features. As we are in the unary predicate setting (which corresponds to predicting an attribute of an object), the counts of the instances of the body of the clause simply means aggregating over the values of the body. This is typically done in several different ways such as mean, weighted mean or noisy-or (Natarajan et al.

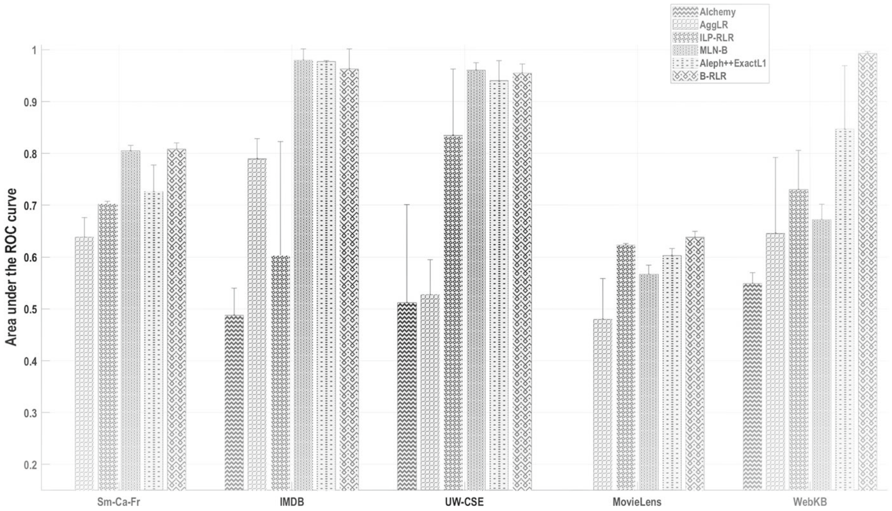

Fig. 3 Comparing the area under the ROC curve for the proposed B- RLR approach to (1) MLN in Alchemy, (2) Propositional logistic regression with relational features (AGG- LR), (3) an approach where rules are learned using a logic learner followed by weight learning (ILP- RLR), (4) a variant of Aleph to learn ILP clauses followed by parameter learning (Aleph++ExactL1) and (5) state-of-the-art MLN with boosted structure learning (MLN- B)

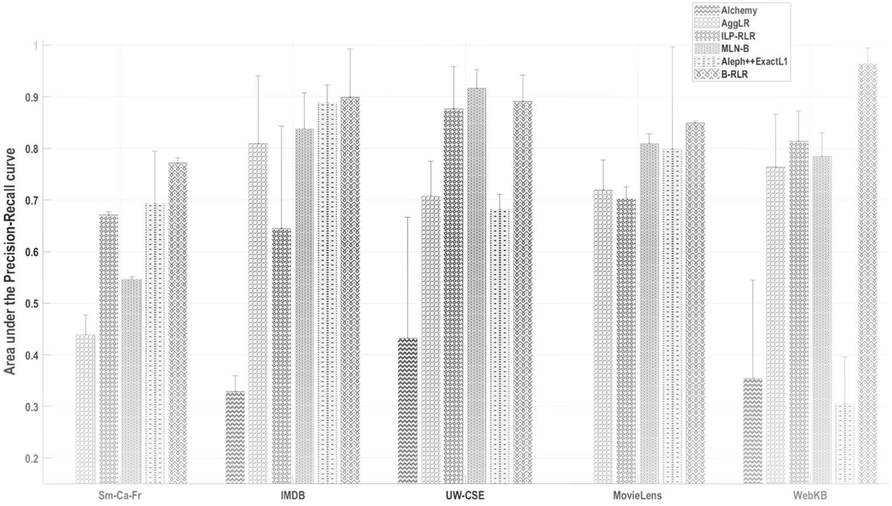

Fig. 4 Comparing the area under the Precision-Recall (PR) curve for the proposed B- RLR approach to (1) MLN in Alchemy, (2) Propositional logistic regression with relational features (AGG- LR), (3) an approach where rules are learned using a logic learner followed by weight learning (ILP- RLR), (4) a variant of Aleph to learn ILP clauses followed by parameter learning ($Aleph + +ExactL1$) and (5) state-of-the-art MLN with boosted structure learning (MLN- B)

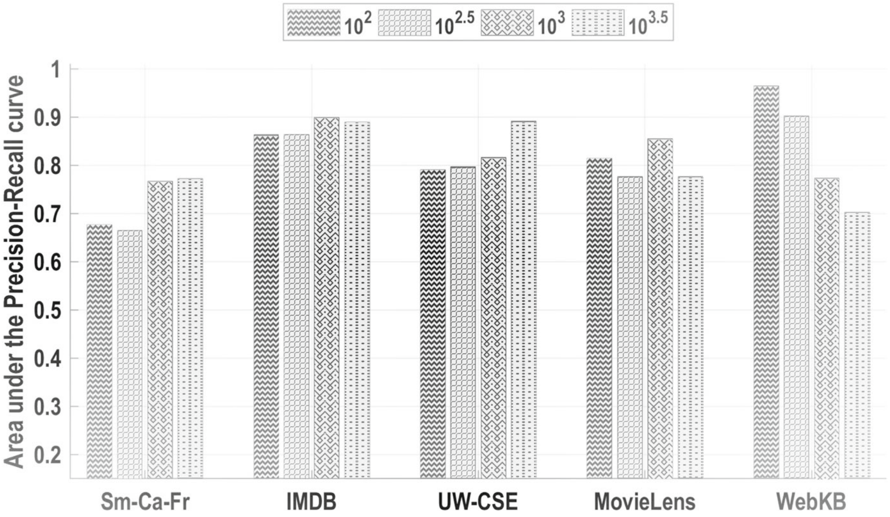

Fig. 5 Sensitivity of the proposed B- RLR approach is analyzed by comparing the Area under the Precision-Recall (PR) curve as λ changes

2008). We suggest the use of logistic function with counts as an alternative aggregator that seems effective in this domain and we hypothesize its use for many relational tasks where aggregation can yield natural models. RLR can effectively model a link prediction task i.e., it can be used to determine whether a relation exists between two objects from their respective properties as well as using other known relations. The formulation of this problem is identical to that of its use in MLNs, with the only difference that the goal is now to infer the value of $C(x, value)$, instead of $R(x_i, x_j)$. The task used in WebKB, MovieLens and Smokes-Cancer-Friends experiments are examples of link prediction. Also, while MLNs only employ counts as their features, RLR allows for a more complex aggregation within the sigmoid function that can use count features in its inner loop. Validating this positive aspect of RLR models remains an interesting future research direction. These results help in answering **Q1** by stating that B- RLR is on par or significantly better than MLN- B in all domains.

Third, from the figures, it can be observed that B- RLR significantly outperforms AGG- LR in several domains. At the outset, this may not be surprising since relational models have been shown to outperform non-relational models. However, the features that are created for the AGG- LR model are the count features of the type defined in the original RLR work and are more expressive than the standard features of propositional models. This result is particularly insightful as the B- RLR model that uses count features, predicates and their combinations themselves in a formula is far more expressive than simple aggregate features. This allows us to answer **Q2** strongly in that the proposed approach is significantly better than an engineered (relational) logistic regression approach.

Finally, comparing the proposed approach to a two-step approach of learning clauses followed by the corresponding weights; B- RLR is significantly better in both PR-space as well as ROC-space than ILP- RLR & ALEPH++EXACTL1 . Hence, **Q3** can be

Table 4 *P*-values for two-tailed *t*-test on AUC-PR for UWCSE

p-value	Aleph++ExactL1	MLN-B	ILP-RLR	AggLR	Alchemy
B-RLR	0.009	0.509	0.773	0.009	0.000
Aleph++ExactL1		0.001	0.010	0.130	0.001
MLN-B			0.236	0.000	0.000
ILP-RLR				0.009	0.000
AggLR					0.001

Table 5 *P*-values for two-tailed *t*-test on AUC-PR for IMDB

p-value	Aleph++ExactL1	MLN-B	ILP-RLR	AggLR	Alchemy
B-RLR	0.148	0.081	0.061	0.020	0.000
Aleph++ExactL1		0.314	0.069	0.412	0.001
MLN-B			0.078	0.115	0.000
ILP-RLR				0.144	0.021
AggLR					0.001

Table 6 *P*-values for two-tailed *t*-test on AUC-PR for Smokes-Cancer-Friends

p-value	Aleph++ExactL1	MLN-B	ILP-RLR	AggLR	Alchemy
B-RLR	0.000	0.004	0.041	0.000	0.000
Aleph++ExactL1		0.030	0.917	0.000	0.001
MLN-B			0.003	0.068	0.000
ILP-RLR				0.004	0.000
AggLR					0.001

answered by stating that B- RLR model outperforms ILP-based two-stage learning in all the regions.

We answer **Q4** using the results from Fig. 5: as long as λ is within the set $\langle 10^2, \ldots, 10^{3.5} \rangle$, our algorithm is not sensitive in most domains. The arguments follow (Yang et al. 2014), and we pick our λ from the specific range using domain knowledge. We pick our λ from this given range for each data set based on a tuning set. We learn on $(n-1)$ folds and test on the remaining fold. It must be mentioned that our parameter λ has a nice intuitive interpretation: they reflect and incorporate the high class imbalance that exists for real-world domains where this is an important practical consideration.

We used paired *t*-test with *p* values = 0.05 for determining the statistical significance. Tables 4, 5, 6, 7, and 8 shows the *t*-test results on the AUC-PR values. From the Figs. 3 and 4, across most domains, we observe that, B- RLR has tighter error bounds compared to the baselines in majority of domains indicating lower variance and subsequently higher generalization performance.

Table 7 P-values for two-tailed t-test on AUC-PR for MovieLens

p-value	Aleph++ExactL1	MLN-B	ILP-RLR	AggLR	Alchemy
B-RLR	0.224	0.014	0.001	0.004	0.000
Aleph++ExactL1		0.802	0.102	0.044	0.001
MLN-B			0.000	0.023	0.000
ILP-RLR				0.546	0.000
AggLR					0.001

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

p-value	Aleph++ExactL1	MLN-B	ILP-RLR	AggLR	Alchemy
B-RLR	0.224	0.014	0.001	0.004	0.000
Aleph++ExactL1		0.005	0.002	0.003	0.005
MLN-B			0.911	0.284	0.008
ILP-RLR				0.696	0.002
AggLR					0.005

Table 9 reports the training time taken in seconds by each method averaged over all the folds in every domain. Timings reported for AGG- LR include time taken for propositional feature construction and weight learning using WEKA tool. For ILP- RLR & $Aleph++ExactL1$, it includes the total time taken to learn rules, count satisfied instances, and learn weights for the rules accordingly. The two boosted approaches timings are reported from the full runs. The results show that the methods are comparable across all the domains - in the domains where boosted methods are faster than the other baselines, grounding of the entire data set caused the increased time for the baselines. Conversely, in the other domains, repeated counting of boosting increased the time in two of the five domains. The results indicate that the proposed $b-rlr$ approach does not sacrifice efficiency (time) for effectiveness (performance).

In summary, our proposed boosted approach appears to be promising across a diversity of relational domains both in terms of effectiveness as well as efficiency.

5 Conclusions

We considered the problem of learning relational logistic regression (RLR) models using the machinery of functional-gradient boosting. To this end, we introduce an alternative interpretation of RLR models that allows us to consider both the true and false groundings of a formula within a single equation. This allowed us to learn vector-weighted clauses that are more compact and expressive compared to standard boosted SRL models. We derived gradients for the different weights, and outlined a learning algorithm that learned first-order features as clauses and the corresponding weights simultaneously. We evaluated the algorithm on standard data sets, and demonstrated the efficacy of the learning algorithm.

Table 9 Comparing the average learning time (in seconds) for the proposed B- RLR approach to (1) standard logistic regression with relational information (AGG- LR), (2) an approach where rules are learned using a logic learner followed by weight learning (ILP- RLR), and (3) state-of-the-art MLN with boosted structure learning (MLN- B)

	Learning Time (seconds)					
	Alchemy	AGG- LR	ILP- RLR	MLN- B	Aleph++ ExactL1	B- RLR
WorkedUnder (IMDB)	1482.63	0.25	18.35	34.94	36.11	54.76
AdvisedBy (UWCS)	105,462	0.58	54.33	92.00	61.09	123.55
Female (movie lens)	–	1.98	22.73	26.63	111.18	66.03
Cancer (SmCaFr)	–	1.06	52.81	603.67	91.23	452.34
Faculty (WebKB)	26,792	0.14	11.89	22.70	55.33	23.44

Learning time includes time to learn the structure, counting the satisfied (or unsatisfied) groundings and weight learning

There are several possible extensions for future work, currently, our method learns a model for a single target predicate deterministically. As mentioned earlier, it is possible to learn a joint model across multiple predicates in a manner akin to learning a relational dependency network (RDN). This can yield a new interpretation for RDNs based on combining rules. Second, learning from truly hybrid data remains a challenge for SRL models in general, and RFGB in particular. Finally, given the recent surge of sequential decision-making research, RLR can be seen as an effective function approximator for relational Markov decision processes (MDPs); employing this novel B- RLR model in the context of relational reinforcement learning can be an exciting and interesting future research direction.

Acknowledgements SN & NR gratefully acknowledge AFOSR award FA9550-18-1-0462 and the support of relationalAI. We thank the anonymous reviewers for their insightful comments and in significantly improving the paper. Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the AFOSR, or the US government.

References

Blockeel H (1999) Top–down induction of first order logical decision trees. AI Commun 12(1–2):119–120

Craven M, McCallum A, PiPasquo D, Mitchell T, Freitag D (1998) Learning to extract symbolic knowledge from the world wide web. Proceedings of the 15th AAAI conference on artificial intelligence, pp 509–516

Dietterich TG, Ashenfelter A, Bulatov Y (2004) Training conditional random fields via gradient tree boosting. In: Proceedings of the twenty–first international conference on machine learning, p 28

Domingos P, Lowd D (2009) Markov logic: an interface layer for artificial intelligence, vol 3. Morgan & Claypool Publishers

Fatemi B, Kazemi SM, Poole D (2016) A learning algorithm for relational logistic regression: preliminary results. arXiv preprint arXiv:1606.08531

Friedman JH (2001) Greedy function approximation: a gradient boosting machine. Ann Stat, pp 1189–1232

Gutmann B, Kersting K (2006) Tildecrf: conditional random fields for logical sequences. In: European conference on machine learning, Springer, Berlin, pp 174–185

Getoor L, Taskar B, eds. (2007) Introduction to statistical relational learning. MIT Press, Cambridge

Harper FM, Konstan JA (2015) The movielens datasets: history and context. ACM Trans Interact Intell Systems 5(4):1–19

Heckerman D, Meek C, Koller D (2007) Probabilistic entity–relationship models, PRMs, and plate models. Introduction to statistical relational learning pp 201–238

Huynh TN, Mooney RJ (2008) Discriminative structure and parameter learning for markov logic networks. In: Proceedings of the 25th international conference on machine learning, pp 416–423

Kazemi SM, Poole D (2018) Relnn: a deep neural model for relational learning. In: Proceedings of the AAAI conference on artificial intelligence, vol 32

Kazemi SM, Buchman D, Kersting K, Natarajan S, Poole D (2014a) Relational logistic regression: the directed analog of markov logic networks. In: AAAI workshop: statistical relational artificial intelligence, Citeseer

Kersting K, De Raedt L (2007) Bayesian logic programming: theory and tool. In: Taskar B, Getoor L (ed). An introduction to statistical relational learning, MIT Press

Kersting K, Driessens K (2008) Non–parametric policy gradients: a unified treatment of propositional and relational domains. In: Proceedings of the 25th international conference on machine learning, pp 456–463

Khot T, Natarajan S, Kersting K, Shavlik J (2011) Learning markov logic networks via functional gradient boosting. In: 2011 IEEE 11th international conference on data mining, IEEE, pp 320–329

Kimmig A, Bach S, Broecheler M, Huang B, Getoor L (2012) A short introduction to probabilistic soft logic. In: Proceedings of the NIPS workshop on probabilistic programming: foundations and applications, pp 1–4

Kok S, Domingos P (2005) Learning the structure of markov logic networks. In: Proceedings of the 22nd international conference on machine learning, pp 441–448

Kok S, Domingos P (2009) Learning markov logic network structure via hypergraph lifting. In: Proceedings of the 26th annual international conference on machine learning, pp 505–512

Koller D (1999) Probabilistic relational models. In: International conference on inductive logic programming, Springer, pp 3–13

Lowd D, Domingos P (2007) Efficient weight learning for markov logic networks. In: European conference on principles of data mining and knowledge discovery, Springer, pp 200–211

Malec M, Khot T, Nagy J, Blasch E, Natarajan S (2016) Inductive logic programming meets relational databases: an application to statistical relational learning

McCullagh P (1984) Generalized linear models. Eur J Oper Res 16(3):285–292

Mihalkova L, Mooney RJ (2007) Bottom–up learning of markov logic network structure. In: Proceedings of the 24th international conference on machine learning, pp 625–632

Muggleton S (1995) Inverse entailment and Progol. New Gener Comput 13(3–4):245–286

Muggleton S (1996) Learning from positive data. In: International conference on inductive logic programming, Springer, pp 358–376

Muggleton S, De Raedt L (1994) Inductive logic programming: theory and methods. J Log Program 19:629–679

Natarajan S, Tadepalli P, Dietterich TG, Fern A (2008) Learning first-order probabilistic models with combining rules. Ann Math Artif Intell 54(1):223–256

Natarajan S, Joshi S, Tadepalli P, Kersting K, Shavlik J (2011) Imitation learning in relational domains: a functional–gradient boosting approach. In: IJCAI proceedings–international joint conference on artificial intelligence, Citeseer, vol 22, p 1414

Natarajan S, Khot T, Kersting K, Gutmann B, Shavlik J (2012) Gradient-based boosting for statistical relational learning: the relational dependency network case. Mach Learn 86(1):25–56

Natarajan S, Prabhakar A, Ramanan N, Bagilone A, Siek K, Connelly K (2017) Boosting for postpartum depression prediction. In: 2017 IEEE/ACM international conference on connected health: applications, systems and engineering technologies (CHASE), IEEE, pp 232–24

Poole D, Buchman D, Kazemi SM, Kersting K, Natarajan S (2014) Population size extrapolation in relational probabilistic modelling. In: International conference on scalable uncertainty management, Springer, pp 292–305

Popescul A, Ungar LH (2003) Structural logistic regression for link analysis. Departmental Papers (CIS) p 133

Raedt LD, Kersting K, Natarajan S, Poole D (2016) Statistical relational artificial intelligence: logic, probability, and computation. Synth Lect Artif Intell Mach Learn 10(2):1–189

Richardson M, Domingos P (2006) Markov logic networks. Mach Learn 62(1–2):107–136

Srinivasan A (2001) The aleph manual. http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/

Taskar B, Abbeel P, Wong MF, Koller D (2007) Relational markov networks. Introduction to statistical relational learning, pp 175–200

Yang S, Khot T, Kersting K, Kunapuli G, Hauser K, Natarajan S (2014) Learning from imbalanced data in relational domains: a soft margin approach. In: 2014 IEEE international conference on data mining, IEEE, pp 1085–1090

Yang S, Korayem M, AlJadda K, Grainger T, Natarajan S (2017) Combining content-based and collaborative filtering for job recommendation system: a cost-sensitive statistical relational learning approach. Knowl-Based Syst 136:37–45