Recursive Random Fields

Daniel Lowd and **Pedro Domingos**
Department of Computer Science and Engineering
University of Washington, Seattle, WA 98195-2350, USA
{lowd,pedrod}@cs.washington.edu

Abstract

A formula in first-order logic can be viewed as a tree, with a logical connective at each node, and a knowledge base can be viewed as a tree whose root is a conjunction. Markov logic [Richardson and Domingos, 2006] makes this conjunction probabilistic, as well as the universal quantifiers directly under it, but the rest of the tree remains purely logical. This causes an asymmetry in the treatment of conjunctions and disjunctions, and of universal and existential quantifiers. We propose to overcome this by allowing the features of Markov logic networks (MLNs) to be nested MLNs. We call this representation *recursive random fields (RRFs)*. RRFs can represent many first-order distributions exponentially more compactly than MLNs. We perform inference in RRFs using MCMC and ICM, and weight learning using a form of backpropagation. Weight learning in RRFs is more powerful than structure learning in MLNs. Applied to first-order knowledge bases, it provides a very flexible form of theory revision. We evaluate RRFs on the problem of probabilistic integrity constraints in databases, and obtain promising results.

1 Introduction

Recent years have seen the development of increasingly powerful combinations of relational and probabilistic representations, along with inference and learning algorithms for them. One of the most general representations to date is Markov logic, which attaches weights to first-order formulas and views them as templates for features of Markov random fields [Richardson and Domingos, 2006]. While Markov logic may be the language of choice for many applications, its unification of logic and probability is incomplete. This is because it only treats the top-level conjunction and universal quantifiers in a knowledge base as probabilistic, when in principle any logical combination can be viewed as the limiting case of an underlying probability distribution. In Markov logic, disjunctions and existential quantifiers remain deterministic. Thus the symmetry between conjunctions and disjunctions, and between universal and existential quantifiers, is lost (except in the infinite-weight limit).

For example, an MLN with the formula $\mathtt{R(X)} \wedge \mathtt{S(X)}$ can treat worlds that violate both $\mathtt{R(X)}$ and $\mathtt{S(X)}$ as less probable than worlds that only violate one. Since an MLN acts as a soft conjunction, the groundings of $\mathtt{R(X)}$ and $\mathtt{S(X)}$ simply appear as distinct formulas. (MLNs convert the knowledge base to CNF before performing learning or inference.) This is not possible for the disjunction $\mathtt{R(X)} \vee \mathtt{S(X)}$: no distinction is made between satisfying both $\mathtt{R(X)}$ and $\mathtt{S(X)}$ and satisfying just one. Since a universally quantified formula is effectively a conjunction over all its groundings, while an existentially quantified formula is a disjunction over them, this leads to the two quantifiers being handled differently.

This asymmetry can be avoided by "softening" disjunction and existential quantification in the same way that Markov logic softens conjunction and universal quantification. The result is a representation where MLNs can have nested MLNs as features. We call these recursive Markov logic networks, or *recursive random fields (RRFs)* for short.

RRFs have many desirable properties, including the ability to represent distributions like noisy DNF, rules with exceptions, and m-of-all quantifiers much more compactly than MLNs. RRFs also allow more flexibilty in revising first-order theories to maximize data likelihood. Standard methods for inference in Markov random fields are easily extended to RRFs, and weight learning can be carried out efficiently using a variant of the backpropagation algorithm.

RRF theory revision can be viewed as a first-order probabilistic analog of the KBANN algorithm, which initializes a neural network with a propositional theory and uses backpropagation to improve its fit to data [Towell and Shavlik, 1994]. A propositional RRF (where all predicates have zero arity) differs from a multilayer perceptron in that its output is the joint probability of its inputs, not the regression of a variable on others (or, in the probabilistic version, its conditional probability). Propositional RRFs are an alternative to Boltzmann machines, with nested features playing the role of hidden variables. Because the nested features are deterministic functions of the inputs, learning does not require EM, and inference does not require marginalizing out variables.

The remainder of this paper is organized as follows. We begin by discussing MLNs and their limitations. We then introduce RRFs along with inference and learning algorithms for them, compare them to MLNs, and present preliminary experimental results.

2 Markov Logic Networks

A Markov logic network (MLN) consists of a set of first-order formulas and weights, $\{(w_i, f_i)\}$, that serve as a template for constructing a Markov random field. Each feature of the Markov random field is a grounding of one of the formulas. The joint probability distribution is therefore:

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

where n_i is the number of true groundings of the ith formula given this assignment, and Z is a normalization constant so that the probabilities of all worlds sum to one. Richardson and Domingos [2006] show that, in finite domains, this generalizes both first-order logic and Markov random fields.

Another way to think about a Markov logic network is as a "softening" of a deterministic knowledge base. A first-order knowledge base can be represented as a conjunction of all groundings of all of its formulas. MLNs relax this conjunction by allowing worlds that violate formulas, but assigning a per-grounding penalty for each violated formula. Worlds that violate many formulas are therefore possible, but less likely than those that violate fewer. In this way, inconsistent knowledge bases can still be useful.

However, while MLNs soften conjunctions and universals, disjunctions and existentials remain deterministic. Just as in MLNs the probability of a world decreases gradually with the number of false groundings of a universally quantified formula, so the probability should increase gradually with the number of true groundings of an existentially quantified formula. RRFs accomplish this.

3 Recursive Random Fields

A recursive random field (RRF) is a log-linear model in which each feature is either an observable random variable or the output of another recursive random field. To build up intuition, we first describe the propositional case, then generalize it to the more interesting relational case. A concrete example is given in Section 3.3, and illustrated in Figure 1.

3.1 Propositional RRFs

While our primary goal is solving relational problems, RRFs may be interesting in propositional domains as well. Propositional RRFs extend Markov random fields and Boltzmann machines in the same way multilayer perceptrons extend single-layer ones. The extension is very simple in principle, but allows RRFs to compactly represent important concepts, such as m-of-n. It also allows RRFs to learn features via weight learning, which could be more effective than current feature-search methods for Markov random fields.

The probability distribution represented by a propositional RRF is as follows:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z_0} \exp\left(\sum_i w_i f_i(\mathbf{x})\right)$$

where Z_0 is a normalization constant, to ensure that the probabilities of all possible states $\mathbf{x}$ sum to 1. What makes this different from a standard Markov random field is that the features can be built up from other subfeatures to an arbitrary number of levels. Specifically, each feature is either:

$$f_i(\mathbf{x}) \;=\; x_j \quad \text{(base case), or}$$

$$f_i(\mathbf{x}) \;=\; \frac{1}{Z_i} \exp\left(\sum_j w_{ij} f_j(\mathbf{x})\right) \quad \text{(recursive case)}$$

In the recursive case, the summation is over all features f_j referenced by the "parent" feature f_i. A child feature, f_j, can appear in more than one parent feature, and thus an RRF can be viewed as a directed acyclic graph of features. The attribute values are at the leaves, and the probability of their configuration is given by the root. (Note that the probabilistic graphical model represented by the RRF is still undirected.)

Since the overall distribution is simply a recursive feature, we can also write the probability distribution as follows:

$$P(\mathbf{X} = \mathbf{x}) = f_0(\mathbf{x})$$

Except for Z_0 (the normalization of the root feature, f_0), the per-feature normalization constants Z_i can be absorbed into the corresponding feature weights w_{ki} in their parent features f_k. Therefore, the user is free to choose any convenient normalization, or even no normalization at all.

It is easy to show that this generalizes Markov random fields with conjunctive or disjunctive features. Each f_i approximates a conjunction when weights w_{ij} are very large. In the limit, f_i will be 1 iff the conjunct is true. f_i can also represent disjuncts using large negative weights, along with a negative weight for the parent feature f_k, w_{ki}. The negative weight w_{ki} turns the conjunction into a disjunction just as negation does in De Morgan's laws. However, one can also move beyond conjunction and disjunction to represent m-of-n concepts, or even more complex distributions where different features have different weights.

Note that features with small absolute weights have little effect. Therefore, instead of using heuristics or search to determine which attributes should appear in which feature, we can include *all* predicates and let weight learning sort out which attributes are relevant for which feature. This is similar to learning a neural network by initializing it with small random values. Since the network can represent any logical formula, there is no need to commit to a specific structure ahead of time. This is an attractive alternative to the traditional inductive methods used for learning MRF features.

An RRF can be seen as a type of multi-layer neural network, in which the node function is exponential (rather than sigmoidal) and the network is trained to maximize joint likelihood. Unlike in multilayer perceptrons, where some random variables are inputs and some are outputs, in RRFs all variables are inputs, and the output is their joint probability. In other ways, an RRF resembles a Boltzmann machine, but with the greater flexibility of multiple layers and learnable using a variant of the back-propagation algorithm. RRFs have no hidden variables to sum out, since all nodes in the network have deterministic values, making inference more efficient.

3.2 Relational RRFs

In the relational case, relations over an arbitrary number of objects take the place of a fixed number of variables. To allow parameter tying across different groundings, we use parameterized features, or *parfeatures*. We represent the parameter tuple as a vector, $\vec{g}$, whose size depends on the arity of the parfeature. Note that $\vec{g}$ is a vector of logical variables (i.e., arguments to predicates) as opposed to the random Boolean variables $\mathbf{x}$ (ground atoms) that represent a state of the world. We use subscripts to distinguish among parfeatures with different parameterizations, e.g. $f_{i,\vec{g}}(\mathbf{x})$ and $f_{i,\vec{g}'}(\mathbf{x})$ represent different groundings of the ith parfeature.

Each RRF parfeature is defined in one of two ways:

$$f_{i,\vec{g}}(\mathbf{x}) = \mathtt{R_i}(\mathtt{g_{i_1}}, \ldots, \mathtt{g_{i_k}}) \quad \text{(base case)}$$

$$f_{i,\vec{g}}(\mathbf{x}) = \frac{1}{Z_i} \exp\left(\sum_j w_{ij} \sum_{\vec{g}'} f_{j,\vec{g},\vec{g}'}(\mathbf{x})\right) \quad \text{(recursive case)}$$

The base case is straightforward: it simply represents the truth value of a ground relation (as specified by $\mathbf{x}$). There is one such grounding for each possible combination of parameters (arguments) of the parfeature. The recursive case sums the weighted values of all child parfeatures. Each parameter $\mathtt{g_i}$ of a child parfeature is either a parameter of the parent feature ($\mathtt{g_i} \in \vec{g}$) or a parameter of a child feature that is summed out and does not appear in the parent feature ($\mathtt{g_i} \in \vec{g}'$). (These $\vec{g}'$ parameters are analogous to the parameters that appear in the body but not the head of a Horn clause.) Just as sums of child features act as conjunctions, the summations over $\vec{g}'$ parameters act as universal quantifiers with Markov logic semantics. In fact, these generalized quantifiers can represent m-of-all concepts, just as the simple feature sums can represent m-of-n concepts.

The relational version of a recursive random field is therefore defined as follows:

$$P(\mathbf{X} = \mathbf{x}) = f_0(\mathbf{x})$$

where $\mathbf{X}$ is the set of all ground relations (e.g., $\mathtt{R(A,B)}$, $\mathtt{S(A)}$), $\mathbf{x}$ is an assignment of truth values to ground relations, and f_0 is the root recursive parfeature (which, being the root, has no parameters). Since f_0 is a recursive parfeature, it is normalized by the constant Z_0 to ensure a valid probability distribution. (As in the propositional case, all other Z_i's can be absorbed into the weights of their parent features, and may therefore be normalized in any convenient way.)

Any relational RRF can be converted into a propositional RRF by grounding all parfeatures and expanding all summations. Each distinct grounding of a parfeature becomes a distinct feature, but with shared weights.

3.3 RRF Example

To clarify these ideas, let us take the example knowledge base from Richardson and Domingos [2006]. The domain consists of three predicates: $\mathtt{Smokes(g)}$ ($\mathtt{g}$ is a smoker); $\mathtt{Cancer(g)}$ ($\mathtt{g}$ has cancer); and $\mathtt{Friends(g,h)}$ ($\mathtt{g}$ is a friend of $\mathtt{h}$). We abbreviate these predicates as $\mathtt{Sm(g)}$, $\mathtt{Ca(g)}$, and $\mathtt{Fr(g,h)}$, respectively.

We wish to represent three beliefs: (i) smoking causes cancer; (ii) friends of friends are friends (transitivity of friendship); and (iii) everyone has at least one friend who smokes. (The most interesting belief from Richardson and Domingos [2006], that people tend to smoke if their friends do, is omitted here for simplicity.) We demonstrate how to represent these beliefs by first converting them to first-order logic, and then converting to an RRF.

One can represent the first belief, "smoking causes cancer," in first-order logic as a universally quantified implication: $\forall \mathtt{g}.\mathtt{Sm(g)} \Rightarrow \mathtt{Ca(g)}$. This implication can be rewritten as a disjunction: $\neg\mathtt{Sm(g)} \vee \mathtt{Ca(g)}$. From De Morgan's laws, this is equivalent to: $\neg(\mathtt{Sm(g)} \wedge \neg\mathtt{Ca(g)})$, which can be represented as an RRF feature:

$$f_{1,\mathtt{g}}(\mathbf{x}) = \frac{1}{Z_1} \exp(w_{1,1}\mathtt{Sm(g)} + w_{1,2}\mathtt{Ca(g)})$$

where $w_{1,1}$ is positive, $w_{1,2}$ is negative, and the feature weight $w_{0,1}$ is negative (not shown above). In general, since RRF features can model conjunction and disjunction, any CNF knowledge base can be represented as an RRF. A similar approach works for the second belief, "friends of people are friends."

The first two beliefs are also handled well by Markov logic networks. The key advantage of recursive random fields is in representing more complex formulas. The third belief, "everyone has at least one friend who smokes," is naturally represented by nested quantifiers: $\forall \mathtt{g}.\exists \mathtt{h}.\mathtt{Fr(g,h)} \wedge \mathtt{Sm(h)}$. This is best represented as an RRF feature that references a secondary feature:

$$f_{3,\mathtt{g}}(\mathbf{x}) = \frac{1}{Z_3} \exp\left(\sum_{\mathtt{h}} w_{3,1} f_{4,\mathtt{g},\mathtt{h}}(\mathbf{x})\right)$$

$$f_{4,\mathtt{g},\mathtt{h}}(\mathbf{x}) = \frac{1}{Z_4} \exp(w_{4,1}\mathtt{Fr(g,h)} + w_{4,2}\mathtt{Sm(h)})$$

Note that in RRFs this feature can also represent a distribution over the number of smoking friends each person has, depending on the assigned weights. It's possible that, while almost everyone has at least one smoking friend, many people have at least two or three. With an RRF, we can actually learn this distribution from data.

This third belief is very problematic for an MLN. First of all, in an MLN it is purely logical: there's no change in probability with the number of smoking friends once that number exceeds one. Secondly, MLNs do not represent the belief efficiently. In an MLN, the existential quantifier is converted to a very large disjunction:

$$(\mathtt{Fr(g,A)} \wedge \mathtt{Sm(A)}) \vee (\mathtt{Fr(g,B)} \wedge \mathtt{Sm(B)}) \vee \cdots$$

If there are 1000 objects in the database, then this disjunction is over 1000 conjunctions. Further, the MLN will convert this DNF into CNF form, leading to 2^{1000} CNF clauses from each grounding of this rule.

These features define a full joint distribution as follows:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z_0} \exp\Big(\sum_{\mathtt{g}} w_{0,1} f_{1,\mathtt{g}}(\mathbf{x}) + \sum_{\mathtt{g,h,i}} w_{0,2} f_{2,\mathtt{g,h,i}}(\mathbf{x}) \sum_{\mathtt{g}} w_{0,3} f_{3,\mathtt{g}}(\mathbf{x})\Big)$$

Figure 1 diagrams the first-order knowledge base containing all of these beliefs, along with the corresponding RRF.

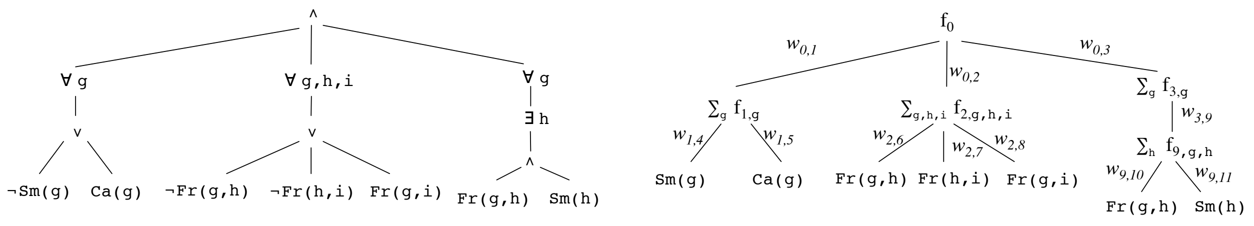

Figure 1: Comparison of first-order logic and RRF structures. The RRF structure closely mirrors that of first-order logic, but connectives and quantifiers are replaced by weighted sums.

4 Inference

Since RRFs generalize MLNs, which in turn generalize finite first-order logic and Markov random fields, exact inference is intractable. Instead, we use MCMC inference, in particular Gibbs sampling. This is straightforward: we sample each unknown ground predicate in turn, conditioned on all other ground predicates. The conditional probability of a particular ground predicate may easily be computed by evaluating the relative probabilities when the predicate is true and when it is false.

We speed this up significantly by caching feature sums. When a predicate is updated, it notifies its parents of the change so that only the necessary values are recomputed.

Our current implementation of MAP inference uses iterated conditional modes (ICM) [Besag, 1986], a simple method for finding a mode of a distribution. Starting from a random configuration, ICM sets each variable in turn to its most likely value, conditioned on all other variables. This procedure continues until no single-variable change will further improve the probability. ICM is easy to implement, fast to run, and guaranteed to converge. Unfortunately, it has no guarantee of converging to the most likely overall configuration. Possible improvements include random restarts, simulated annealing, etc.

We also use ICM to find an initial state for the Gibbs sampler. By starting at a mode, we significantly reduce the burn-in time and achieve better predictions sooner.

5 Learning

Given a particular RRF structure and initial set of weights, we learn weights using a novel variant of the back-propagation algorithm. As in traditional back-propagation, the goal is to efficiently compute the derivative of the loss function with respect to each weight in the model. In this case, the loss function is not the error in predicting the output variables, but rather the joint log likelihood of all variables. We must also consider the partition function for the root feature, Z_0. For these computations, we extract the $1/Z_0$ term from f_0, and use f_0 refer to the unnormalized feature value.

We begin by discussing the simpler, propositional case. We abbreviate $f_i(\mathbf{x})$ as f_i for these arguments. The derivative of the log likelihood with respect to a weight w_{ij} consists of two terms:

$$\frac{\partial \log P(\mathbf{x})}{\partial w_{ij}} = \frac{\partial \log(f_0/Z_0)}{\partial w_{ij}} = \frac{\partial \log(f_0)}{\partial w_{ij}} - \frac{\partial \log(Z_0)}{\partial w_{ij}}$$

The first term can be evaluated with the chain rule:

$$\frac{\partial \log(f_0)}{\partial w_{ij}} = \frac{\partial \log(f_0)}{\partial f_i}\frac{\partial f_i}{\partial w_{ij}}$$

From the definition of f_i (including the normalization Z_i):

$$\frac{\partial f_i}{\partial w_{ij}} = f_i\left(f_j - \frac{1}{Z_i}\frac{\partial Z_i}{\partial w_{ij}}\right)$$

From repeated applications of the chain rule, the $\partial \log(f_0)/\partial f_i$ term is the sum of all derivatives along all paths through the network from f_0 to f_i. Given a path in the feature graph $\{f_0, f_a, \ldots, f_k, f_i\}$, the derivative along that path takes the form $f_0 w_a f_a w_b f_b \cdots w_k f_k w_i$. We can efficiently compute the sum of all paths by caching the per-feature partials, $\partial f_0/\partial f_i$, analogous to back-propagation.

The second term, $\partial \log(Z_0)/\partial w_{ij}$, is the expected value of the first term, evaluated over all possible inputs $\mathbf{x}'$. Therefore, the complete partial derivative is:

$$\frac{\partial \log P(\mathbf{x})}{\partial w_{ij}} = \frac{\partial \log(f_0(\mathbf{x}))}{\partial w_{ij}} - E_{\mathbf{x}'}\left[\frac{\partial \log(f_0(\mathbf{x}'))}{\partial w_{ij}}\right]$$

where the individual components are evaluated as above.

Computing the expectation is typically intractable, but it can be approximated using Gibbs sampling. A more efficient alternative, used by Richardson and Domingos [2006], is to instead optimize the pseudo-likelihood (P^*) [Besag, 1975]:

$$P^*(\mathbf{X}{=}\mathbf{x}) = \prod_{t=1}^{n} P(X_t = x_t | MB_x(X_t))$$

where $MB_x(X_t)$ is the state of the Markov blanket of X_t in the data. Pseudo-likelihood is a consistent estimator, but little else is known about its formal properties. It can perform poorly when long chains of inference are required, but worked quite well in our test domain.

The expression for the gradient of the pseudo-log-likelihood of a propositional RRF is as follows:

$$\begin{aligned}\frac{\partial \log P^*(\mathbf{X}{=}\mathbf{x})}{\partial w_i} &= \sum_{t=1}^{n} P(X_t = \neg x_t | MB_x(X_t)) \\ &\quad \times \left(\frac{\partial \log f_0}{\partial w_i} - \frac{\partial \log f_{0[X_t=\neg x_t]}}{\partial w_i}\right)\end{aligned}$$

We can compute this by iterating over all query predicates, toggling each one in turn, and computing the relative likelihood and unnormalized likelihood gradient for that permuted state. Note that we compute the gradient of the unnormalized log likelihood as a subroutine in computing the gradient of the pseudo-log-likelihood. However, we no longer need to approximate the intractable normalization term, Z.

To learn a relational RRF, we use the domain to instantiate a propositional RRF with tied weights. The number of features as well as the number of children per feature will depend on the number of objects in the domain. Instead of a weight being attached to a single feature, it is now attached to a set of groundings of a parfeature. The partial derivative with respect to a weight is therefore the sum of the partial derivatives with respect to each instantiation of the shared weight.

6 RRFs vs. MLNs

Both RRFs and MLNs subsume probabilistic models and first-order logic in finite domains. Both can be trained generatively or discriminatively using gradient descent, either to optimize log likelihood or pseudo-likelihood. For both, when optimizing log likelihood, the normalization constant Z_0 can be approximated using the most probable explanation or MCMC.

Any MLN can be converted into a relational RRF by translating each clause into an equivalent parfeature. With sufficiently large weights, a parfeature approximates a hard conjunction or disjunction over its children. However, when its weights are sufficiently distinct, a parfeature can take on a different value for each configuration of its children. This allows RRFs to compactly represent distributions that would require an exponential number of clauses in an MLN.

Any RRF can be converted to an MLN by flattening the model, but this will typically require an exponential number of clauses. Such an MLN would be intractable for learning or inference. RRFs are therefore much better at modeling soft disjunction, existential quantification, and nested formulas.

In addition to being "softer" than an MLN clause, an RRF parfeature can represent many different MLN clauses simply by adjusting its weights. This makes RRF weight learning more powerful than MLN structure learning: an RRF with $n + 1$ recursive parfeatures (one for the root) can represent any MLN structure with up to n clauses, as well as many distributions that an n-clause MLN cannot represent.

This leads to new alternatives for structure learning and theory revision. In a domain where little background knowledge is available, an RRF could be initialized with small random weights and still converge to a good statistical model. This is potentially much better than MLN structure learning, which constructs clauses one predicate at a time, and must adjust weights to evaluate every candidate clause.

When background knowledge is available, we can begin by initializing the RRF to the background theory, just as in MLNs. However, in addition to the known dependencies, we can also add dependencies on other parfeatures or predicates with very small weights. Weight learning can learn large weights for relevant dependencies and negligible weights for irrelevant dependencies. This is analagous to what the KBANN system does using neural networks [Towell and Shavlik, 1994]. In contrast, MLNs can only do theory revision through discrete search.

7 Experiments: Probabilistic Integrity Constraints

Integrity constraints are statements in first-order logic that are used to detect and repair database errors [Abiteboul *et al.*, 1995]. Logical statements work well when errors are few and critical, but are increasingly impractical with noisy databases such as those that arise from the integration of multiple databases, information extraction from the web, etc. We want to make these constraints *probabilistic*, so we can statistically infer the types of errors and their sources. To date, there has been very little work on this problem. (See [Andritsos *et al.*, 2006; Ilyas *et al.*, 2004] for two early approaches.) This is a natural domain for MLNs and RRFs, since both use first-order formulas to construct probability distributions over worlds, or databases.

The two most common types of integrity constraints are inclusion constraints and functional dependencies. Inclusion constraints are of the form:

$$\forall \mathtt{x}.(\exists \mathtt{y}.\mathtt{R}(\mathtt{x},\mathtt{y})) \Rightarrow (\exists \mathtt{z}.\mathtt{S}(\mathtt{x},\mathtt{z}))$$

For example, in Company X, `R` could be the relation "`ProjectLead`" (`x` is in charge of project `y`) and `S` could be the relation "`ManagerOf`" (`x` manages employee `z`). This constraint says that every project leader manages at least one other worker. Of course, some employees could manage other employees without being the lead on any project.

To evaluate MLNs and RRFs on inclusion constraints, we generated domains consisting of 100 people and 100 projects. With probability 0.25, a person `x` is a project leader, and leads (or co-leads) each project `y` with probability 0.1. For each project `x` leads, `x` manages employee `z` with probability 0.05. Additional managing relationships are generated with a probability that we vary from 0.001 to 0.1. However, the actual leadership and management relationships are unobserved: we only see noisy versions, which are corrupted with a probability that we vary from 0.0 to 1.0.

We converted the constraint formula into an MLN and an RRF as described in previous sections and added the implications $\mathtt{ProjectLead}(\mathtt{x},\mathtt{y}) \Rightarrow \mathtt{ProjectLead}'(\mathtt{x},\mathtt{y})$ and $\mathtt{ManagerOf}(\mathtt{x},\mathtt{z}) \Rightarrow \mathtt{ManagerOf}'(\mathtt{x},\mathtt{z})$, where the predicates with primes are the observed ones. We found that both MLNs and RRFs worked better when given both directions of the integrity constraint, since project leaders manage people and managers lead projects. We learned weights to optimize pseudo-log-likelihood in all models. The results are shown in Figure 2. Each data point is an average over 10 train/test set pairs. RRFs show a consistent advantage over MLNs, because they can better represent the fact that an employee who manages many people is probably a project leader, while an employee who manages few people may not be.

The second type of integrity constraints, functional dependencies, are of the form:

$$\forall \mathtt{x},\mathtt{y}_1,\mathtt{y}_2.(\exists \mathtt{z}_1,\mathtt{z}_2.\mathtt{R}(\mathtt{x},\mathtt{y}_1,\mathtt{z}_1) \wedge \mathtt{R}(\mathtt{x},\mathtt{y}_2,\mathtt{z}_2)) \Rightarrow \mathtt{y}_1 = \mathtt{y}_2$$

In a functional dependency, each `x` determines a unique `y` (or an equivalence set of `y`s). For example, suppose Company X has a table of parts suppliers represented by the relation $\mathtt{Supplier}(\mathtt{TaxID},\mathtt{CompanyName},\mathtt{PartType})$. A sup-

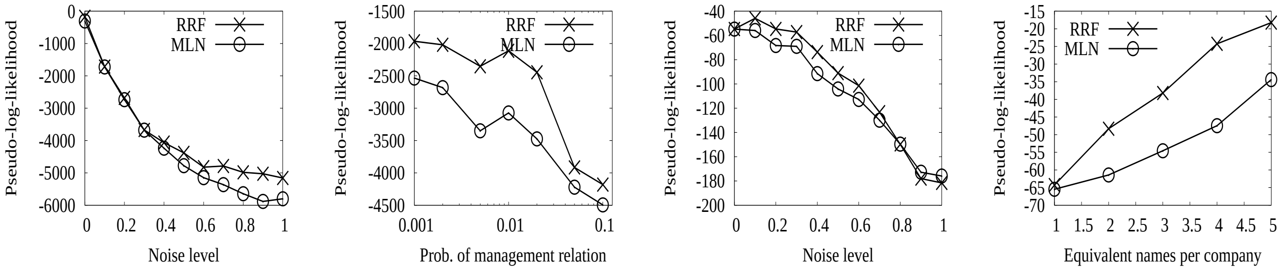

Figure 2: Log-pseudo-likelihood of RRFs and MLNs for the inclusion constraints and functional dependencies. For inclusion constraints, we vary the probability of corrupting each observed relation, $\texttt{ProjectLead}'(\texttt{x},\texttt{y})$ and $\texttt{Manages}'(\texttt{x},\texttt{z})$ (far left) and the probability of adding extra $\texttt{Manages}(\texttt{x},\texttt{z})$ tuples (mid-left). For functional dependencies, we vary the probability of corrupting the company name of a supplier (mid-right) and the number of equivalent names per company (far right).

plier that supplies multiple types of parts may appear multiple times, but each tax ID should always be associated with the same company name or set of equivalent company names (e.g., "Microsoft," "Microsoft, Inc.," and "MSFT" are all equivalent). If there is noise in the database, logic alone cannot say which company names are equivalent and which are errors.

For evaluating functional dependencies, we generated a database with 30 tax IDs, 30 company names (partitioned into equivalence sets of size k), and 30 part types. Each tax ID is associated with one set of equivalent company names, uniformly selected from the 30/k name groups. For each company name a tax ID uses, we generated part types z that the company supplies with probability 0.25. This simulates the merging of k distinct databases, each with its own company naming scheme but with shared identifiers for the other fields. As a final step, we randomly corrupted company names with a probability that we varied from 0.0 to 1.0.

The task was to predict which pairs of company names were equivalent, given the supplier table. The results are shown in Figure 2. With multiple databases at moderate levels of noise, RRFs outperform MLNs. We attribute this to RRFs' ability to represent the fact that two company names are more likely to be equivalent if they both appear multiple times with the same tax ID, and to learn the appropriate form of this dependence.

8 Conclusion

Recursive random fields overcome some salient limitations of Markov logic. While MLNs only model uncertainty over conjunctions and universal quantifiers, RRFs also model uncertainty over disjunctions and existentials, and thus achieve a deeper integration of logic and probability. Inference in RRFs can be carried out using Gibbs sampling and iterated conditional modes, and weights can be learned using a variant of the back-propagation algorithm.

The main disadvantage of RRFs relative to MLNs is reduced understandability. One possibility is to extract MLNs from RRFs with techniques similar to those used to extract propositional theories from KBANN models. Another important problem for future work is scalability. Here we plan to adapt many of the MLN optimizations to RRFs.

Most importantly, we intend to apply RRFs to real datasets to better understand how they work in practice, and to see if their greater representational power yields better models.

Acknowledgments

This work was supported by a National Science Foundation Graduate Research Fellowship awarded to the first author, DARPA grant FA8750-05-2-0283 (managed by AFRL), DARPA contract NBCH-D030010, NSF grant IIS-0534881, and ONR grant N00014-05-1-0313. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of DARPA, NSF, ONR, or the United States Government.

References

[Abiteboul *et al.*, 1995] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[Andritsos *et al.*, 2006] P. Andritsos, A. Fuxman, and R. Miller. Clean answers over dirty databases: A probabilistic approach. In *Proc. ICDE-06*, page 30, 2006

[Besag, 1975] J. Besag. Statistical analysis of non-lattice data. *The Statistician*, 24:179-195, 1975.

[Besag, 1986] J. Besag. On the statistical analysis of dirty pictures. *J. Royal Stat. Soc. B*, 48:259–302, 1986.

[Ilyas *et al.*, 2004] I. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: Automatic discovery of correlations and soft functional dependencies. In *Proc. SIGMOD-04*, pages 647–658, 2004.

[Richardson and Domingos, 2006] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.

[Towell and Shavlik, 1994] G. Towell and J. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70:119–165, 1994.