12 Markov Logic: A Unifying Framework for Statistical Relational Learning

Pedro Domingos and Matthew Richardson

Interest in statistical relational learning (SRL) has grown rapidly in recent years. Several key SRL tasks have been identified, and a large number of approaches have been proposed. Increasingly, a unifying framework is needed to facilitate transfer of knowledge across tasks and approaches, to compare approaches, and to help bring structure to the field. We propose *Markov logic* as such a framework. Syntactically, Markov logic is indistinguishable from first-order logic, except that each formula has a weight attached. Semantically, a set of Markov logic formulae represents a probability distribution over possible worlds, in the form of a log-linear model with one feature per grounding of a formula in the set, with the corresponding weight. We show how approaches like probabilistic relational models, knowledge-based model construction, and stochastic logic programs can be mapped into Markov logic. We also show how tasks like collective classification, link prediction, link-based clustering, social network modeling, and object identification can be concisely formulated in Markov logic. Finally, we develop learning and inference algorithms for Markov logic, and report experimental results on a link prediction task.

12.1 The Need for a Unifying Framework

Many (if not most) real-world application domains are characterized by the presence of both uncertainty and complex relational structure. Statistical learning focuses on the former, and relational learning on the latter. Statistical relational learning (SRL) seeks to combine the power of both. Research in SRL has expanded rapidly in recent years, both because of the need for it in applications, and because statistical and relational learning have individually matured to the point where combining them is a feasible research enterprise. A number of key SRL tasks have been identified, including collective classification, link prediction, link-based clustering, social network modeling, object identification, and others. A large and growing number of

SRL approaches have been proposed, including knowledge-based model construction [55, 39, 29], stochastic logic programs [37, 9], PRISM [51], MACCENT [12], probabilistic relational models [17], relational Markov models [1], relational Markov networks [53], relational dependency networks [38], structural logistic regression [44], relational generation functions [7], constraint logic programming for probabilistic knowledge (CLP($\mathcal{BN}$)) [50], and others.

While the variety of problems and approaches in the field is valuable, it makes it difficult for researchers, students, and practitioners to identify, learn, and apply the essentials. In particular, for the most part, the relationships between different approaches and their relative strengths and weaknesses remain poorly understood, and innovations in one task or application do not easily transfer to others, slowing down progress. There is thus an increasingly pressing need for a unifying framework, a common language for describing and relating the different tasks and approaches. To be most useful, such a framework should satisfy the following desiderata:

1. *The framework must incorporate both first-order logic and probabilistic graphical models.* Otherwise some current or future SRL approaches will fall outside its scope.
2. *SRL problems should be representable clearly and simply in the framework.*
3. *The framework must facilitate the use of domain knowledge in SRL.* Because the search space for SRL algorithms is very large even by AI standards, domain knowledge is critical to success. Conversely, the ability to incorporate rich domain knowledge is one of the most attractive features of SRL.
4. *The framework should facilitate the extension to SRL of techniques from statistical learning, inductive logic programming, probabilistic inference, and logical inference.* This will speed progress in SRL by taking advantage of the large extant literature in these areas.

In this chapter we propose *Markov logic* as a framework that we believe meets all of these desiderata. We begin by briefly reviewing the necessary background in Markov networks (section 12.2) and first-order logic (section 12.3). We then introduce Markov logic (section 12.4) and describe how several SRL approaches and tasks can be formulated in this framework (sections 12.5 and 12.6). Next, we show how techniques from logic, probabilistic inference, statistics and inductive logic programming can be used to obtain practical inference and learning algorithms for Markov logic (sections 12.7 and 12.8). Finally, we illustrate the application of these algorithms in a real-world link prediction task (section 12.9) and conclude (section 12.10).

12.2 Markov Networks

A *Markov network* (also known as a *Markov random field*) is a model for the joint distribution of a set of variables $X = (X_1, X_2, \ldots, X_n) \in \mathcal{X}$ [41]. It is composed of an undirected graph G and a set of potential functions ϕ_k. The graph has a node for each variable, and the model has a potential function for each clique in the graph. A potential function is a non-negative real-valued function of the state of the corresponding clique. The joint distribution represented by a Markov network is given by

$$P(X = x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}}), \tag{12.1}$$

where $x_{\{k\}}$ is the state of the kth clique (i.e., the state of the variables that appear in that clique). Z, known as the *partition function*, is given by $Z = \sum_{x \in \mathcal{X}} \prod_k \phi_k(x_{\{k\}})$. Markov networks are often conveniently represented as *log-linear models*, with each clique potential replaced by an exponentiated weighted sum of features of the state, leading to

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_j w_j f_j(x) \right). \tag{12.2}$$

A feature may be any real-valued function of the state. This chapter will focus on binary features, $f_j(x) \in \{0, 1\}$. In the most direct translation from the potential-function form (12.1), there is one feature corresponding to each possible state $x_{\{k\}}$ of each clique, with its weight being $\log \phi_k(x_{\{k\}})$. This representation is exponential in the size of the cliques. However, we are free to specify a much smaller number of features (e.g., logical functions of the state of the clique), allowing for a more compact representation than the potential-function form, particularly when large cliques are present. Markov Login Networks (MLNs) will take advantage of this.

Inference in Markov networks is #P-complete [49]. The most widely used method for approximate inference in Markov networks is Markov chain Monte Carlo (MCMC) [20], and in particular Gibbs sampling, which proceeds by sampling each variable in turn given its Markov blanket. (The Markov blanket of a node is the minimal set of nodes that renders it independent of the remaining network; in a Markov network, this is simply the node's neighbors in the graph.) Marginal probabilities are computed by counting over these samples; conditional probabilities are computed by running the Gibbs sampler with the conditioning variables clamped to their given values. Another popular method for inference in Markov networks is belief propagation [57].

Maximum likelihood or maximup a posteriori (MAP) estimates of Markov network weights cannot be computed in closed form, but, because the log-likelihood is a concave function of the weights, they can be found efficiently using standard gradient-based or quasi-Newton optimization methods [40]. Another alternative is iterative scaling [13]. Features can also be learned from data, by, for example, greedily constructing conjunctions of atomic features [13].

12.3 First-Order Logic

A *first-order knowledge base (KB)* is a set of sentences or formulae in first-order logic [18]. Formulae are constructed using four types of symbols: constants, variables, functions, and predicates. Constant symbols represent objects in the domain of interest (e.g., people: `Anna`, `Bob`, `Chris`, etc.). Variable symbols range over the objects in the domain. Function symbols (e.g., `MotherOf`) represent mappings from tuples of objects to objects. Predicate symbols represent relations among objects in the domain (e.g., `Friends`) or attributes of objects (e.g., `Smokes`). An *interpretation* specifies which objects, functions, and relations in the domain are represented by which symbols. Variables and constants may be *typed*, in which case variables range only over objects of the corresponding type, and constants can only represent objects of the corresponding type. For example, the variable `x` might range over people (e.g., Anna, Bob, etc.), and the constant `C` might represent a city (e.g, Seattle, Tokyo, etc.).

A *term* is any expression representing an object in the domain. It can be a constant, a variable, or a function applied to a tuple of terms. For example, `Anna`, `x`, and `GreatestCommonDivisor(x, y)` are terms. An *atomic formula* or *atom* is a predicate symbol applied to a tuple of terms (e.g., `Friends(x, MotherOf(Anna))`). Formulae are recursively constructed from atomic formulae using logical connectives and quantifiers. If F_1 and F_2 are formulae, the following are also formulae: $\neg F_1$ (negation), which is true iff F_1 is false; $F_1 \wedge F_2$ (conjunction), which is true iff both F_1 and F_2 are true; $F_1 \vee F_2$ (disjunction), which is true iff F_1 or F_2 is true; $F_1 \Rightarrow F_2$ (implication), which is true iff F_1 is false or F_2 is true; $F_1 \Leftrightarrow F_2$ (equivalence), which is true iff F_1 and F_2 have the same truth-value; $\forall \mathtt{x}\ F_1$ (universal quantification), which is true iff F_1 is true for every object `x` in the domain; and $\exists \mathtt{x}\ F_1$ (existential quantification), which is true iff F_1 is true for at least one object `x` in the domain. Parentheses may be used to enforce precedence. A *positive literal* is an atomic formula; a *negative literal* is a negated atomic formula. The formulae in a KB are implicitly conjoined, and thus a KB can be viewed as a single large formula. A *ground term* is a term containing no variables. A *ground atom* or *ground predicate* is an atomic formula all of whose arguments are ground terms. A *possible world* or *Herbrand interpretation* assigns a truth value to each possible ground predicate.

A formula is *satisfiable* iff there exists at least one world in which it is true. The basic inference problem in first-order logic is to determine whether a knowledge base KB *entails* a formula F, i.e., if F is true in all worlds where KB is true (denoted by $KB \models F$). This is often done by *refutation*: KB entails F iff $KB \cup \neg F$ is unsatisfiable. (Thus, if a KB contains a contradiction, all formulae trivially follow from it, which makes painstaking knowledge engineering a necessity.) For automated

Table 12.1 Example of a first-order knowledge base and MLN. `Fr()` is short for `Friends()`, `Sm()` for `Smokes()`, and `Ca()` for `Cancer()`

English	First-order logic	Clausal form	Wt
Friends of friends are friends	$\forall x \forall y \forall z\, \mathtt{Fr}(x,y) \wedge \mathtt{Fr}(y,z) \Rightarrow \mathtt{Fr}(x,z)$	$\neg\mathtt{Fr}(x,y) \vee \neg\mathtt{Fr}(y,z) \vee \mathtt{Fr}(x,z)$	0.7
Friendless people smoke.	$\forall x\, (\neg(\exists y\, \mathtt{Fr}(x,y)) \Rightarrow \mathtt{Sm}(x))$	$\mathtt{Fr}(x,g(x)) \vee \mathtt{Sm}(x)$	2.3
Smoking causes cancer.	$\forall x\, \mathtt{Sm}(x) \Rightarrow \mathtt{Ca}(x)$	$\neg\mathtt{Sm}(x) \vee \mathtt{Ca}(x)$	1.5
If two people are friends, either both smoke or neither does.	$\forall x \forall y\, \mathtt{Fr}(x,y) \Rightarrow (\mathtt{Sm}(x) \Leftrightarrow \mathtt{Sm}(y))$	$\neg\mathtt{Fr}(x,y) \vee \mathtt{Sm}(x) \vee \neg\mathtt{Sm}(y)$, $\neg\mathtt{Fr}(x,y) \vee \neg\mathtt{Sm}(x) \vee \mathtt{Sm}(y)$	1.1 1.1

inference, it is often convenient to convert formulae to a more regular form, typically *clausal form* (also known as *conjunctive normal form (CNF)*). A KB in clausal form is a conjunction of *clauses*, a clause being a disjunction of literals. Every KB in first-order logic can be converted to clausal form using a mechanical sequence of steps.[1] Clausal form is used in resolution, a sound and refutation-complete inference procedure for first-order logic [48].

Inference in first-order logic is only semidecidable. Because of this, knowledge bases are often constructed using a restricted subset of first-order logic with more desirable properties. The most widely used restriction is to *Horn clauses*, which are clauses containing at most one positive literal. The Prolog programming language is based on Horn clause logic [34]. Prolog programs can be learned from databases by searching for Horn clauses that (approximately) hold in the data; this is studied in the field of inductive logic programming (ILP) [32].

Table 12.1 shows a simple KB and its conversion to clausal form. Notice that, while these formulae may be *typically* true in the real world, they are not *always* true. In most domains it is very difficult to come up with non-trivial formulae that are always true, and such formulae capture only a fraction of the relevant knowledge. Thus, despite its expressiveness, pure first-order logic has limited applicability to practical AI problems. Many ad hoc extensions to address this have been proposed. In the more limited case of propositional logic, the problem is well solved by probabilistic graphical models. The next section describes a way to generalize these models to the first-order case.

1. This conversion includes the removal of existential quantifiers by Skolemization, which is not sound in general. However, in finite domains an existentially quantified formula can simply be replaced by a disjunction of its groundings.

12.4 Markov Logic

A first-order KB can be seen as a set of hard constraints on the set of possible worlds: if a world violates even one formula, it has zero probability. The basic idea in Markov logic is to soften these constraints: when a world violates one formula in the KB it is less probable, but not impossible. The fewer formulae a world violates, the more probable it is. Each formula has an associated weight that reflects how strong a constraint it is: the higher the weight, the greater the difference in log probability between a world that satisfies the formula and one that does not, other things being equal. We call a set of formulae in Markov logic a *Markov logic network*. MLNs define probability distributions over possible worlds [21] as follows.

Definition 12.1
An MLN L is a set of pairs (F_i, w_i), where F_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, \ldots, c_{|C|}\}$, it defines a Markov network $M_{L,C}$ ((12.1) and (12.2)) as follows:

1. $M_{L,C}$ contains one binary node for each possible grounding of each predicate appearing in L. The value of the node is 1 if the ground atom is true, and 0 otherwise.
2. $M_{L,C}$ contains one feature for each possible grounding of each formula F_i in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the w_i associated with F_i in L.

The syntax of the formulae in an MLN is the standard syntax of first-order logic [18]. Free (unquantified) variables are treated as universally quantified at the outermost level of the formula.

An MLN can be viewed as a *template* for constructing Markov networks. Given different sets of constants, it will produce different networks, and these may be of widely varying size, but all will have certain regularities in structure and parameters, given by the MLN (e.g., all groundings of the same formula will have the same weight). We call each of these networks a *ground Markov network* to distinguish it from the first-order MLN. From definition 12.1 and (12.1) and (12.2), the probability distribution over possible worlds x specified by the ground Markov network $M_{L,C}$ is given by

$$P(X=x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right) = \frac{1}{Z} \prod_i \phi_i(x_{\{i\}})^{n_i(x)}. \tag{12.3}$$

where $n_i(x)$ is the number of true groundings of F_i in x, $x_{\{i\}}$ is the state (truth values) of the atoms appearing in F_i, and $\phi_i(x_{\{i\}}) = e^{w_i}$. Notice that, although we defined MLNs as log-linear models, they could equally well be defined as products of potential functions, as the second equality above shows. This will be the most convenient approach in domains with a mixture of hard and soft constraints (i.e.,

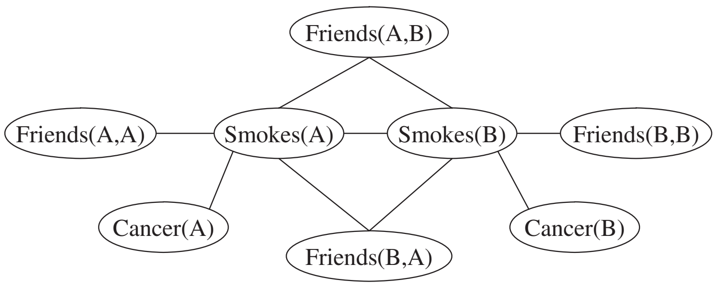

Figure 12.1 Ground Markov network obtained by applying the last two formulae in table 12.1 to the constants `Anna`(`A`) and `Bob`(`B`).

where some formulae hold with certainty, leading to zero probabilities for some worlds).

The graphical structure of $M_{L,C}$ follows from definition 12.1: there is an edge between two nodes of $M_{L,C}$ iff the corresponding ground atoms appear together in at least one grounding of one formula in L. Thus, the atoms in each ground formula form a (not necessarily maximal) clique in $M_{L,C}$. Figure 12.1 shows the graph of the ground Markov network defined by the last two formulae in table 12.1 and the constants `Anna` and `Bob`. Each node in this graph is a ground atom (e.g., `Friends(Anna, Bob)`). The graph contains an arc between each pair of atoms that appear together in some grounding of one of the formulae. $M_{L,C}$ can now be used to infer the probability that Anna and Bob are friends given their smoking habits, the probability that Bob has cancer given his friendship with Anna and whether she has cancer, etc.

Each state of $M_{L,C}$ represents a possible world. A possible world is a set of objects, a set of functions (mappings from tuples of objects to objects), and a set of relations that hold between those objects; together with an interpretation, they determine the truth-value of each ground atom. The following assumptions ensure that the set of possible worlds for (L, C) is finite, and that $M_{L,C}$ represents a unique, well-defined probability distribution over those worlds, irrespective of the interpretation and domain. These assumptions are quite reasonable in most practical applications, and greatly simplify the use of MLNs. For the remaining cases, we discuss below the extent to which each one can be relaxed.

Assumption 1
Unique names Different constants refer to different objects [18].

Assumption 2
Domain closure The only objects in the domain are those representable using the constant and function symbols in (L, C) [18].

Assumption 3
Known functions For each function appearing in L, the value of that function applied to every possible tuple of arguments is known, and is an element of C.

Table 12.2 Construction of all groundings of a first-order formula under assumptions 1–3

```
function Ground($F$, $C$)
    inputs: $F$, a formula in first-order logic
            $C$, a set of constants
    output: $G_F$, a set of ground formulae
    calls: $CNF(F, C)$, which converts $F$ to conjunctive normal form, replacing
           existentially quantified formulae by disjunctions of their groundings over $C$
$F \leftarrow CNF(F, C)$
$G_F = \emptyset$
for each clause $F_j \in F$
    $G_j = \{F_j\}$
    for each variable $x$ in $F_j$
        for each clause $F_k(x) \in G_j$
            $G_j \leftarrow (G_j \setminus F_k(x)) \cup \{F_k(c_1), F_k(c_2), \ldots, F_k(c_{|C|})\}$,
                where $F_k(c_i)$ is $F_k(x)$ with $x$ replaced by $c_i \in C$
    $G_F \leftarrow G_F \cup G_j$
for each ground clause $F_j \in G_F$
    repeat
        for each function $f(a_1, a_2, \ldots)$ all of whose arguments are constants
            $F_j \leftarrow F_j$ with $f(a_1, a_2, \ldots)$ replaced by $c$, where $c = f(a_1, a_2, \ldots)$
    until $F_j$ contains no functions
return $G_F$
```

This last assumption allows us to replace functions by their values when grounding formulae. Thus the only ground atoms that need to be considered are those having constants as arguments. The infinite number of terms constructible from all functions and constants in (L, C) (the Herbrand universe of (L, C)) can be ignored, because each of those terms corresponds to a known constant in C, and atoms involving them are already represented as the atoms involving the corresponding constants. The possible groundings of a predicate in definition 12.1 are thus obtained simply by replacing each variable in the predicate with each constant in C, and replacing each function term in the predicate by the corresponding constant. Table 12.2 shows how the groundings of a formula are obtained given assumptions 1–3. If a formula contains more than one clause, its weight is divided equally among the clauses, and a clause's weight is assigned to each of its groundings.

Assumption 1 (unique names) can be removed by introducing the equality predicate ($\mathtt{Equals(x, y)}$, or $\mathtt{x} = \mathtt{y}$ for short) and adding the necessary axioms to the MLN: equality is reflexive, symmetric, and transitive; for each unary predicate $\mathtt{P}$, $\forall \mathtt{x} \forall \mathtt{y}\, \mathtt{x} = \mathtt{y} \Rightarrow (\mathtt{P(x)} \Leftrightarrow \mathtt{P(y)})$; and similarly for higher-order predicates and functions [18]. The resulting MLN will have a node for each pair of constants, whose value is 1 if the constants represent the same object and 0 otherwise; these nodes will be connected to each other and to the rest of the network by arcs representing the axioms above. Notice that this allows us to make probabilistic inferences about the equality of two constants. We have successfully used this as the basis of an approach to object identification (see section 12.6.5).

If the number u of unknown objects is known, assumption 2 (domain closure) can be removed simply by introducing u arbitrary new constants. If u is unknown but finite, assumption 2 can be removed by introducing a distribution over u, grounding the MLN with each number of unknown objects, and computing the probability of a formula F as $P(F) = \sum_{u=0}^{u_{max}} P(u)P(F|M^u_{L,C})$, where $M^u_{L,C}$ is the ground MLN with u unknown objects. An infinite u requires extending MLNs to the case $|C| = \infty$.

Let $H_{L,C}$ be the set of all ground terms constructible from the function symbols in L and the constants in L and C (the "Herbrand universe" of (L, C)). Assumption 3 (known functions) can be removed by treating each element of $H_{L,C}$ as an additional constant and applying the same procedure used to remove the unique names assumption. For example, with a function `G(x)` and constants `A` and `B`, the MLN will now contain nodes for `G(A)` = `A`, `G(A)` = `B`, etc. This leads to an infinite number of new constants, requiring the corresponding extension of MLNs. However, if we restrict the level of nesting to some maximum, the resulting MLN is still finite.

To summarize, assumptions 1–3 can be removed as long as the domain is finite. We believe it is possible to extend MLNs to infinite domains (see Jaeger [27]), but this is an issue of chiefly theoretical interest, and we leave it for future work. In the remainder of this chapter we proceed under assumptions 1–3, except where noted.

A first-order KB can be transformed into an MLN simply by assigning a weight to each formula. For example, the clauses and weights in the last two columns of Table 12.1 constitute an MLN. According to this MLN, other things being equal, a world where n friendless people are nonsmokers is $e^{(2.3)n}$ times less probable than a world where all friendless people smoke. Notice that all the formulae in table 12.1 are false in the real world as universally quantified logical statements, but capture useful information on friendships and smoking habits, when viewed as features of a Markov network. For example, it is well-known that teenage friends tend to have similar smoking habits [35]. In fact, an MLN like the one in table 12.1 succinctly represents a type of model that is a staple of social network analysis [54].

It is easy to see that MLNs subsume essentially all propositional probabilistic models, as detailed below.

Proposition 12.2
Every probability distribution over discrete or finite-precision numeric variables can be represented as a Markov logic network.

Proof Consider first the case of Boolean variables $(X_1, X_2, \ldots, X_n)$. Define a predicate of zero arity R_h for each variable X_h, and include in the MLN L a formula for each possible state of $(X_1, X_2, \ldots, X_n)$. This formula is a conjunction of n literals, with the hth literal being $R_h()$ if X_h is true in the state, and $\neg R_h()$ otherwise. The formula's weight is $\log P(X_1, X_2, \ldots, X_n)$. (If some states have zero probability, use instead the product form (see [12.3]), with $\phi_i()$ equal to the probability of the ith state.) Since all predicates in L have zero arity, L defines the same Markov network $M_{L,C}$ irrespective of C, with one node for each variable X_h.

For any state, the corresponding formula is true and all others are false, and thus (12.3) represents the original distribution (notice that $Z = 1$). The generalization to arbitrary discrete variables is straightforward, by defining a zero-arity predicate for each value of each variable. Similarly for finite-precision numeric variables, by noting that they can be represented as Boolean vectors. ∎

Of course, compact factored models like Markov networks and Bayesian networks can still be represented compactly by MLNs, by defining formulae for the corresponding factors (arbitrary features in Markov networks, and states of a node and its parents in Bayesian networks).[2]

First-order logic (with assumptions 1–3 above) is the special case of Markov logic obtained when all weights are equal and tend to infinity, as described below.

Proposition 12.3
Let KB be a satisfiable knowledge base, L be the MLN obtained by assigning weight w to every formula in KB, C be the set of constants appearing in KB, $P_w(x)$ be the probability assigned to a (set of) possible world(s) x by $M_{L,C}$, $\mathcal{X}_{KB}$ be the set of worlds that satisfy KB, and F be an arbitrary formula in first-order logic. Then:

1. $\forall x \in \mathcal{X}_{KB}\ \lim_{w\to\infty} P_w(x) = |\mathcal{X}_{KB}|^{-1}$
   $\forall x \notin \mathcal{X}_{KB}\ \lim_{w\to\infty} P_w(x) = 0$
2. For all F, $KB \models F$ iff $\lim_{w\to\infty} P_w(F) = 1$

Proof Let k be the number of ground formulae in $M_{L,C}$. By (12.3), if $x \in \mathcal{X}_{KB}$, then $P_w(x) = e^{kw}/Z$, and if $x \notin \mathcal{X}_{KB}$ then $P_w(x) \leq e^{(k-1)w}/Z$. Thus all $x \in \mathcal{X}_{KB}$ are equiprobable and $\lim_{w\to\infty} P(\mathcal{X} \setminus \mathcal{X}_{KB})/P(\mathcal{X}_{KB}) \leq \lim_{w\to\infty}(|\mathcal{X} \setminus \mathcal{X}_{KB}|/|\mathcal{X}_{KB}|)e^{-w} = 0$, proving part 1. By definition of entailment, $KB \models F$ iff every world that satisfies KB also satisfies F. Therefore, letting $\mathcal{X}_F$ be the set of worlds that satisfies F, if $KB \models F$, then $\mathcal{X}_{KB} \subseteq \mathcal{X}_F$ and $P_w(F) = \sum_{x\in\mathcal{X}_F} P_w(x) \geq P_w(\mathcal{X}_{KB})$. Since, from part 1, $\lim_{w\to\infty} P_w(\mathcal{X}_{KB}) = 1$, this implies that if $KB \models F$, then $\lim_{w\to\infty} P_w(F) = 1$. The inverse direction of part 2 is proved by noting that if $\lim_{w\to\infty} P_w(F) = 1$, then every world with nonzero probability must satisfy F, and this includes every world in $\mathcal{X}_{KB}$. ∎

In other words, in the limit of all equal infinite weights, the MLN represents a uniform distribution over the worlds that satisfy the KB, and all entailment queries can be answered by computing the probability of the query formula and checking whether it is 1. Even when weights are finite, first-order logic is "embedded" in Markov logic in the following sense. Assume without loss of generality that all weights are non-negative. (A formula with a negative weight w can be replaced by its negation with weight $-w$.) If the KB composed of the formulae in an

2. While some conditional independence structures can be compactly represented with directed graphs but not with undirected ones, they still lead to compact models in the form of Equation 12.3 (i.e., as products of potential functions).

MLN L (negated, if their weight is negative) is satisfiable, then, for any C, the satisfying assignments are the modes of the distribution represented by $M_{L,C}$. This is because the modes are the worlds x with maximum $\sum_i w_i n_i(x)$ (see [12.3]), and this expression is maximized when all groundings of all formulae are true (i.e., the KB is satisfied). Unlike an ordinary first-order KB, however, an MLN can produce useful results even when it contains contradictions. An MLN can also be obtained by merging several KBs, even if they are partly incompatible. This is potentially useful in areas like the Semantic Web [2] and mass collaboration [46].

It is interesting to see a simple example of how Markov logic generalizes first-order logic. Consider an MLN containing the single formula $\forall \mathtt{x}\ \mathtt{R(x)} \Rightarrow \mathtt{S(x)}$ with weight w, and $C = \{\mathtt{A}\}$. This leads to four possible worlds: $\{\neg\mathtt{R(A)}, \neg\mathtt{S(A)}\}$, $\{\neg\mathtt{R(A)}, \mathtt{S(A)}\}$, $\{\mathtt{R(A)}, \neg\mathtt{S(A)}\}$, and $\{\mathtt{R(A)}, \mathtt{S(A)}\}$. From (12.3) we obtain that $P(\{\mathtt{R(A)}, \neg\mathtt{S(A)}\}) = 1/(3e^w + 1)$ and the probability of each of the other three worlds is $e^w/(3e^w + 1)$. (The denominator is the partition function Z; see section 12.2.) Thus, if $w > 0$, the effect of the MLN is to make the world that is inconsistent with $\forall \mathtt{x}\ \mathtt{R(x)} \Rightarrow \mathtt{S(x)}$ less likely than the other three. From the probabilities above we obtain that $P(\mathtt{S(A)}|\mathtt{R(A)}) = 1/(1 + e^{-w})$. When $w \to \infty$, $P(\mathtt{S(A)}|\mathtt{R(A)}) \to 1$, recovering the logical entailment.

In practice, we have found it useful to add each predicate to the MLN as a unit clause. In other words, for each predicate $R(x_1, x_2, \ldots)$ appearing in the MLN, we add the formula $\forall x_1, x_2, \ldots\ R(x_1, x_2, \ldots)$ with some weight w_R. The weight of a unit clause can (roughly speaking) capture the marginal distribution of the corresponding predicate, leaving the weights of the non-unit clauses free to model only dependencies between predicates.

When manually constructing an MLN or interpreting a learned one, it is useful to have an intuitive understanding of the weights. The weight of a formula F is simply the log odds between a world where F is true and a world where F is false, other things being equal. However, if F shares variables with other formulae, as will typically be the case, it may not be possible to keep the truth-values of those formulae unchanged while reversing F's. In this case there is no longer a one-to-one correspondence between weights and probabilities of formulae.[3] Nevertheless, the probabilities of all formulae collectively determine all weights, if we view them as constraints on a maximum entropy distribution, or treat them as empirical probabilities and learn the maximum likelihood weights (the two are equivalent) [13]. Thus a good way to set the weights of an MLN is to write down the probability with which each formula should hold, treat these as empirical frequencies, and learn the weights from them using the algorithm in section 12.8. Conversely, the weights

3. This is an unavoidable side effect of the power and flexibility of Markov networks. In Bayesian networks, parameters are probabilities, but at the cost of greatly restricting the ways in which the distribution may be factored. In particular, potential functions must be conditional probabilities, and the directed graph must have no cycles. The latter condition is particularly troublesome to enforce in relational extensions [53].

in a learned MLN can be viewed as collectively encoding the empirical formula probabilities.

The size of ground Markov networks can be vastly reduced by having typed constants and variables, and only grounding variables to constants of the same type. However, even in this case the size of the network may be extremely large. Fortunately, many inferences do not require grounding the entire network, as we will see in section 12.7.

12.5 SRL Approaches

Because of the simplicity and generality of Markov logic, many representations used in SRL can be easily mapped into it. In this section, we informally do this for a representative sample of these approaches. The goal is not to capture all of their many details, but rather to help bring structure to the field. Further, converting these representations to Markov logic brings a number of new capabilities and advantages, and we also discuss these.

12.5.1 Knowledge-Based Model Construction

Knowledge-based model construction (KBMC) is a combination of logic programming and Bayesian networks [55, 39, 29]. As in Markov logic, nodes in KBMC represent ground predicates. Given a Horn KB, KBMC answers a query by finding all possible backward-chaining proofs of the query and evidence predicates from each other, constructing a Bayesian network over the ground predicates in the proofs, and performing inference over this network. The parents of a predicate node in the network are deterministic AND nodes representing the bodies of the clauses that have that node as head. The conditional probability of the node given these is specified by a combination function (e.g., noisy OR, logistic regression, arbitrary conditional probability table (CPT)). Markov logic generalizes KBMC by allowing arbitrary formulas (not just Horn clauses) and inference in any direction. It also sidesteps the thorny problem of avoiding cycles in the Bayesian networks constructed by KBMC, and obviates the need for ad hoc combination functions for clauses with the same consequent.

A KBMC model can be translated into Markov logic by writing down a set of formulae for each first-order predicate `Pk`(...) in the domain. Each formula is a conjunction containing `Pk`(...) and one literal per parent of `Pk`(...) (i.e., per first-order predicate appearing in a Horn clause having `Pk`(...) as the consequent). A subset of these literals are negated; there is one formula for each possible combination of positive and negative literals. The weight of the formula is $w = \log[p/(1-p)]$, where p is the conditional probability of the child predicate when the corresponding conjunction of parent literals is true, according to the combination function used. If the combination function is logistic regression, it can be represented using only a linear number of formulae, taking advantage of the fact that a logistic regression model is a (conditional) Markov network with a binary clique between each predictor and the response. Noisy OR can similarly be represented with a linear number of parents.

12.5.2 Other Logic Programming Approaches

Stochastic logic programs (SLPs) [37, 9] are a combination of logic programming and log-linear models. Puech and Muggleton [45] showed that SLPs are a special case of KBMC, and thus they can be converted into Markov logic in the same way. Like Markov logic, SLPs have one coefficient per clause, but they represent distributions over Prolog proof trees rather than over predicates; the latter have to be obtained by marginalization. Similar remarks apply to a number of other representations that are essentially equivalent to SLPs, like independent choice logic [43] and PRISM [51].

MACCENT [12] is a system that learns log-linear models with first-order features; each feature is a conjunction of a class and a Prolog query (clause with empty head). A key difference between MACCENT and Markov logic is that MACCENT is a classification system (i.e., it predicts the conditional distribution of an object's class given its properties), while an MLN represents the full joint distribution of a set of predicates. Like any probability estimation approach, Markov logic can be used for classification simply by issuing the appropriate conditional queries.[4] In particular, a MACCENT model can be converted into Markov logic simply by defining a class predicate (as in section 12.6.1), adding the corresponding features and their weights to the MLN, and adding a formula with infinite weight stating that each object must have exactly one class. (This fails to model the marginal distribution of the nonclass predicates, which is not a problem if only classification queries will be issued.) MACCENT can make use of deterministic background knowledge in the form of Prolog clauses; these can be added to the MLN as formulae with infinite weight. In addition, Markov logic allows uncertain background knowledge (via formulae with finite weights). As described in Subsection 12.6.1, MLNs can be used for collective classification, where the classes of different objects can depend on each other; MACCENT, which requires that each object be represented in a separate Prolog KB, does not have this capability.

Constraint logic programming is an extension of logic programming where variables are constrained instead of being bound to specific values during inference [31]. Probabilistic CLP generalizes SLPs to CLP [47], and CLP($\mathcal{BN}$) combines CLP with Bayesian networks [50]. Unlike in Markov logic, constraints in CLP($\mathcal{BN}$) are hard (i.e., they cannot be violated; rather, they define the form of the probability distribution).

4. Conversely, joint distributions can be built up from classifiers (e.g., [23]), but this would be a significant extension of MACCENT.

12.5.3 Probabilistic Relational Models

Probabilistic relational models (PRMs) [17] are a combination of frame-based systems and Bayesian networks. PRMs can be converted into Markov logic by defining a predicate $\mathtt{S(x,v)}$ for each (propositional or relational) attribute of each class, where $\mathtt{S(x,v)}$ means "The value of attribute S in object x is v." A PRM is then translated into an MLN by writing down a formula for each line of each (class-level) CPT and value of the child attribute. The formula is a conjunction of literals stating the parent values and a literal stating the child value, and its weight is the logarithm of $P(x|Parents(x))$, the corresponding entry in the CPT. In addition, the MLN contains formulae with infinite weight stating that each attribute must take exactly one value. This approach handles all types of uncertainty in PRMs (attribute, reference, and existence uncertainty).

As Taskar et al. [53] point out, the need to avoid cycles in PRMs causes significant representational and computational difficulties. Inference in PRMs is done by creating the complete ground network, which limits their scalability. PRMs require specifying a complete conditional model for each attribute of each class, which in large complex domains can be quite burdensome. In contrast, Markov logic creates a complete joint distribution from whatever number of first-order features the user chooses to specify.

12.5.4 Relational Markov Networks

Relational Markov networks (RMNs) use conjunctive database queries as clique templates [53]. They do not provide a language for defining features. As a result, by default RMNs require a feature for every possible state of a clique, making them exponential in clique size and limiting the complexity of dependencies they can model. Markov logic provides first-order logic as a powerful language for specifying features. Specifying the features also indirectly specifies the cliques, which can be very large as long as the number of relevant features (i.e., formulae) is tractable. Additionally, Markov logic generalizes RMNs by allowing uncertainty over arbitrary relations (not just attributes of individual objects). RMNs are trained discriminatively, and do not specify a complete joint distribution for the variables in the model. Discriminative training of MLNs is straightforward [52]. RMNs use MAP estimation with belief propagation for inference, which makes learning quite slow, despite the simplified discriminative setting; both pseudo-likelihood optimization and the discriminative training described in Singla and Domingos [52] are presumably much faster. To date, no structure-learning algorithms for RMNs have been proposed. MLN structure can be learned using standard inductive logic programming (ILP) techniques, as described later in this chapter, or by directly optimizing pseudo-likelihood, as described in Kok and Domingos [30].

12.5.5 Structural Logistic Regression

In structural logistic regression (SLR) [44], the predictors are the output of SQL queries over the input data. In the same way that a logistic regression model can be viewed as a discriminatively trained Markov network, an SLR model can be viewed as a a discriminatively trained MLN.[5]

12.5.6 Relational Dependency Networks

In a relational dependency network (RDN), each node's probability conditioned on its Markov blanket is given by a decision tree [38]. Every RDN has a corresponding MLN in the same way that every dependency network has a corresponding Markov network, given by the stationary distribution of a Gibbs sampler operating on it [23].

12.5.7 Plates and Probabilistic Entity Relationship Models

Large graphical models with repeated structure are often compactly represented using plates [4]. Markov logic allows plates to be specified using universal quantification. In addition, it allows individuals and their relations to be explicitly represented (see Cussens [8]), and context-specific independences to be compactly written down, instead of left implicit in the node models. More recently, Heckerman et al. [24] have proposed probabilistic entity relationship (ER) models, a language based on ER models that combines the features of plates and PRMs; this language can be mapped into Markov logic in the same way that ER models can be mapped into first-order logic. Probabilistic ER models allow logical expressions as constraints on how ground networks are constructed, but the truth-values of these expressions have to be known in advance; Markov logic allows uncertainty over all logical expressions.

12.5.8 BLOG

Milch et al. [36] have proposed a language, called BLOG (Bayesian Logic), designed to avoid making the unique names and domain closure assumptions. A BLOG program specifies procedurally how to generate a possible world, and does not allow arbitrary first-order knowledge to be easily incorporated. Also, it only specifies the structure of the model, leaving the parameters to be specified by external calls. BLOG models are directed graphs and need to avoid cycles, which substantially complicates their design. We saw in the previous section how to remove the unique names and domain closure assumptions in Markov logic. (When there are unknown objects of multiple types, a random variable for the number of each

5. Use of SQL aggregates requires that their definitions be imported into Markov logic.

type is introduced.) Inference about an object's attributes, rather than those of its observations, can be done simply by having variables for objects as well as for their observations (e.g., for books as well as citations to them). To date, no learning algorithms or practical inference algorithms for BLOG have been proposed.

12.6 SRL Tasks

Many SRL tasks can be concisely formulated in Markov logic, making it possible to see how they relate to each other, and to develop algorithms that are simultaneously applicable to all. In this section we exemplify this with five key tasks: collective classification, link prediction, link-based clustering, social network modeling, and object identification.

12.6.1 Collective Classification

The goal of ordinary classification is to predict the class of an object given its attributes. Collective classification also takes into account the classes of related objects (e.g., [6, 53, 38]). Attributes can be represented in Markov logic as predicates of the form $\mathtt{A}(\mathtt{x}, \mathtt{v})$, where $\mathtt{A}$ is an attribute, $\mathtt{x}$ is an object, and $\mathtt{v}$ is the value of $\mathtt{A}$ in $\mathtt{x}$. The class is a designated attribute $\mathtt{C}$, representable by $\mathtt{C}(\mathtt{x}, \mathtt{v})$, where $\mathtt{v}$ is $\mathtt{x}$'s class. Classification is now simply the problem of inferring the truth-value of $\mathtt{C}(\mathtt{x}, \mathtt{v})$ for all $\mathtt{x}$ and $\mathtt{v}$ of interest given all known $\mathtt{A}(\mathtt{x}, \mathtt{v})$. Ordinary classification is the special case where $\mathtt{C}(\mathtt{x_i}, \mathtt{v})$ and $\mathtt{C}(\mathtt{x_j}, \mathtt{v})$ are independent for all $\mathtt{x_i}$ and $\mathtt{x_j}$ given the known $\mathtt{A}(\mathtt{x}, \mathtt{v})$. In collective classification, the Markov blanket of $\mathtt{C}(\mathtt{x_i}, \mathtt{v})$ includes other $\mathtt{C}(\mathtt{x_j}, \mathtt{v})$, even after conditioning on the known $\mathtt{A}(\mathtt{x}, \mathtt{v})$. Relations between objects are represented by predicates of the form $\mathtt{R}(\mathtt{x_i}, \mathtt{x_j})$. A number of interesting generalizations are readily apparent; for example, $\mathtt{C}(\mathtt{x_i}, \mathtt{v})$ and $\mathtt{C}(\mathtt{x_j}, \mathtt{v})$ may be indirectly dependent via unknown predicates, possibly including the $\mathtt{R}(\mathtt{x_i}, \mathtt{x_j})$ predicates themselves.

12.6.2 Link Prediction

The goal of link prediction is to determine whether a relation exists between two objects of interest (e.g., whether Anna is Bob's Ph.D. advisor) from the properties of those objects and possibly other known relations (e.g., see Popescul and Ungar [44]). The formulation of this problem in Markov logic is identical to that of collective classification, with the only difference that the goal is now to infer the value of $\mathtt{R}(\mathtt{x_i}, \mathtt{x_j})$ for all object pairs of interest, instead of $\mathtt{C}(\mathtt{x}, \mathtt{v})$. The task used in our experiments is an example of link prediction (see section 12.9).

12.6.3 Link-Based Clustering

The goal of clustering is to group together objects with similar attributes. In model-based clustering, we assume a generative model $P(X) = \sum_C P(C)\ P(X|C)$, where X is an object, C ranges over clusters, and $P(C|X)$ is X's degree of membership in cluster C. In link-based clustering, objects are clustered according to their links (e.g., objects that are more closely related are more likely to belong to the same cluster), and possibly according to their attributes as well (e.g., see Flake et al. [16]). This problem can be formulated in Markov logic by postulating an unobserved predicate $\texttt{C}(\texttt{x}, \texttt{v})$ with the meaning "$\texttt{x}$ belongs to cluster $\texttt{v}$," and having formulas in the MLN involving this predicate and the observed ones (e.g., $\texttt{R}(\texttt{x}_\texttt{i}, \texttt{x}_\texttt{j})$ for links and $\texttt{A}(\texttt{x}, \texttt{v})$ for attributes). Link-based clustering can now be performed by learning the parameters of the MLN, and cluster memberships are given by the probabilities of the $\texttt{C}(\texttt{x}, \texttt{v})$ predicates conditioned on the observed ones.

12.6.4 Social Network Modeling

Social networks are graphs where nodes represent social actors (e.g., people) and arcs represent relations between them (e.g., friendship). Social network analysis [54] is concerned with building models relating actors' properties and their links. For example, the probability of two actors forming a link may depend on the similarity of their attributes, and conversely two linked actors may be more likely to have certain properties. These models are typically Markov networks, and can be concisely represented by formulas like $\forall \texttt{x} \forall \texttt{y} \forall \texttt{v}\ \texttt{R}(\texttt{x}, \texttt{y}) \Rightarrow (\texttt{A}(\texttt{x}, \texttt{v}) \Leftrightarrow \texttt{A}(\texttt{y}, \texttt{v}))$, where $\texttt{x}$ and $\texttt{y}$ are actors, $\texttt{R}(\texttt{x}, \texttt{y})$ is a relation between them, $\texttt{A}(\texttt{x}, \texttt{v})$ represents an attribute of $\texttt{x}$, and the weight of the formula captures the strength of the correlation between the relation and the attribute similarity. For example, a model stating that friends tend to have similar smoking habits can be represented by the formula $\forall \texttt{x} \forall \texttt{y}\ \texttt{Friends}(\texttt{x}, \texttt{y}) \Rightarrow (\texttt{Smokes}(\texttt{x}) \Leftrightarrow \texttt{Smokes}(\texttt{y}))$ (table 12.1). As well as encompassing existing social network models, Markov logic allows richer ones to be easily stated (e.g., by writing formulas involving multiple types of relations and multiple attributes, as well as more complex dependencies between them).

12.6.5 Object Identification

Object identification (also known as record linkage, deduplication, and others) is the problem of determining which records in a database refer to the same real-world entity (e.g., which entries in a bibliographic database represent the same publication) [56]. This problem is of crucial importance to many companies, government agencies, and large-scale scientific projects. One way to represent it in Markov logic is by removing the unique names assumption as described in section 12.4, i.e., by defining a predicate $\texttt{Equals}(\texttt{x}, \texttt{y})$ (or $\texttt{x} = \texttt{y}$ for short) with the meaning "$\texttt{x}$ represents the same real-world entity as $\texttt{y}$." This predicate is applied both to records and their fields (e.g., "ICML" = "Intl. Conf. on Mach. Learn."). The dependencies between record matches and field matches can then be represented by formulas like $\forall \mathtt{x} \forall \mathtt{y} \ \ \mathtt{x} = \mathtt{y} \Leftrightarrow \mathtt{f_i}(\mathtt{x}) = \mathtt{f_i}(\mathtt{y})$, where x and y are records and $\mathtt{f_i}(\mathtt{x})$ is a function returning the value of the ith field of record x. We have successfully applied this approach to deduplicating the Cora database of computer science papers [52]. Because it allows information to propagate from one match decision (i.e., one grounding of $\mathtt{x} = \mathtt{y}$) to another via fields that appear in both pairs of records, it effectively performs collective object identification, and in our experiments outperformed the traditional method of making each match decision independently of all others. For example, matching two references may allow us to determine that "ICML" and "MLC" represent the same conference, which in turn may help us to match another pair of references where one contains "ICML" and the other "MLC." Markov logic also allows additional information to be incorporated into a deduplication system easily, modularly, and uniformly. For example, transitive closure is incorporated by adding the formula $\forall \mathtt{x} \forall \mathtt{y} \forall \mathtt{z} \ \ \mathtt{x} = \mathtt{y} \wedge \mathtt{y} = \mathtt{z} \Rightarrow \mathtt{x} = \mathtt{z}$, with a weight that can be learned from data.

12.7 Inference

We now show how inference in Markov logic can be carried out. Markov logic can answer arbitrary queries of the form "What is the probability that formula F_1 holds given that formula F_2 does?" If F_1 and F_2 are two formulae in first-order logic, C is a finite set of constants including any constants that appear in F_1 or F_2, and L is an MLN, then

$$\begin{aligned} P(F_1|F_2, L, C) &= P(F_1|F_2, M_{L,C}) \\ &= \frac{P(F_1 \wedge F_2|M_{L,C})}{P(F_2|M_{L,C})} \\ &= \frac{\sum_{x \in \mathcal{X}_{F_1} \cap \mathcal{X}_{F_2}} P(X{=}x|M_{L,C})}{\sum_{x \in \mathcal{X}_{F_2}} P(X{=}x|M_{L,C})}, \end{aligned} \tag{12.4}$$

where $\mathcal{X}_{F_i}$ is the set of worlds where F_i holds, and $P(x|M_{L,C})$ is given by (12.3). Ordinary conditional queries in graphical models are the special case of (12.4) where all predicates in F_1, F_2, and L are zero-arity and the formulae are conjunctions. The question of whether a knowledge base KB entails a formula F in first-order logic is the question of whether $P(F|L_{KB}, C_{KB,F}) = 1$, where L_{KB} is the MLN obtained by assigning infinite weight to all the formulae in KB, and $C_{KB,F}$ is the set of all constants appearing in KB or F. The question is answered by computing $P(F|L_{KB}, C_{KB,F})$ by (12.4), with $F_2 = \text{True}$.

Computing (12.4) directly will be intractable in all but the smallest domains. Since Markov logic inference subsumes probabilistic inference, which is #P-complete, and logical inference in finite domains, which is NP-complete, no better results can be expected. However, many of the large number of techniques for

Table 12.3 Network construction for inference in Markov logic

```
function ConstructNetwork($F_1, F_2, L, C$)
    inputs: $F_1$, a set of ground atoms with unknown truth-values (the "query")
            $F_2$, a set of ground atoms with known truth-values (the "evidence")
            $L$, a Markov logic network
            $C$, a set of constants
    output: $M$, a ground Markov network
    calls: $MB(q)$, the Markov blanket of $q$ in $M_{L,C}$
$G \leftarrow F_1$
while $F_1 \neq \emptyset$
    for all $q \in F_1$
        if $q \notin F_2$
            $F_1 \leftarrow F_1 \cup (MB(q) \setminus G)$
            $G \leftarrow G \cup MB(q)$
        $F_1 \leftarrow F_1 \setminus \{q\}$
return $M$, the ground Markov network composed of all nodes in $G$, all arcs between
    them in $M_{L,C}$, and the features and weights on the corresponding cliques
```

efficient inference in either case are applicable to Markov logic. Because Markov logic allows fine-grained encoding of knowledge, including context-specific independences, inference in it may in some cases be more efficient than inference in an ordinary graphical model for the same domain. On the logic side, the probabilistic semantics of Markov logic allows for approximate inference, with the corresponding potential gains in efficiency.

In principle, $P(F_1|F_2, L, C)$ can be approximated using an MCMC algorithm that rejects all moves to states where F_2 does not hold, and counts the number of samples in which F_1 holds. However, even this is likely to be too slow for arbitrary formulae. Instead, we provide an inference algorithm for the case where F_1 and F_2 are conjunctions of ground literals. While less general than (12.4), this is the most frequent type of query in practice, and the algorithm we provide answers it far more efficiently than a direct application of (12.4). Investigating lifted inference (where queries containing variables are answered without grounding them) is an important direction for future work (see Jaeger [26] and Poole [42] for initial results). The algorithm proceeds in two phases, analogous to knowledge-based model construction [55]. The first phase returns the minimal subset M of the ground Markov network required to compute $P(F_1|F_2, L, C)$. The algorithm for this is shown in table 12.3. The size of the network returned may be further reduced, and the algorithm sped up, by noticing that any ground formula which is made true by the evidence can be ignored, and the corresponding arcs removed from the network. In the worst case, the network contains $O(|C|^a)$ nodes, where a is the largest predicate arity in the domain, but in practice it may be much smaller.

The second phase performs inference on this network, with the nodes in F_2 set to their values in F_2. Our implementation uses Gibbs sampling, but any inference method may be employed. The basic Gibbs step consists of sampling one ground atom given its Markov blanket. The Markov blanket of a ground atom is the set of ground predicates that appear in some grounding of a formula with it. The probability of a ground atom X_l when its Markov blanket B_l is in state b_l is

$$P(X_l = x_l | B_l = b_l) = \frac{\exp(\sum_{f_i \in F_l} w_i f_i(X_l = x_l, B_l = b_l))}{\exp(\sum_{f_i \in F_l} w_i f_i(X_l = 0, B_l = b_l)) + \exp(\sum_{f_i \in F_l} w_i f_i(X_l = 1, B_l = b_l))}, \tag{12.5}$$

where F_l is the set of ground formulae that X_l appears in, and $f_i(X_l = x_l, B_l = b_l)$ is the value (0 or 1) of the feature corresponding to the ith ground formula when $X_l = x_l$ and $B_l = b_l$. For sets of atoms of which exactly one is true in any given world (e.g., the possible values of an attribute), blocking can be used (i.e., one atom is set to true and the others to false in one step, by sampling conditioned on their collective Markov blanket). The estimated probability of a conjunction of ground literals is simply the fraction of samples in which the ground literals are true, after the Markov chain has converged. Because the distribution is likely to have many modes, we run the Markov chain multiple times. When the MLN is in clausal form, we minimize burn-in time by starting each run from a mode found using MaxWalkSat, a local search algorithm for the weighted satisfiability problem (i.e., finding a truth assignment that maximizes the sum of weights of satisfied clauses) [28]. When there are hard constraints (clauses with infinite weight), MaxWalkSat finds regions that satisfy them, and the Gibbs sampler then samples from these regions to obtain probability estimates.

12.8 Learning

We learn MLN weights from one or more relational databases. (For brevity, the treatment below is for one database, but the generalization to many is trivial.) We make a closed-world assumption [18]: if a ground atom is not in the database, it is assumed to be false. If there are n possible ground atoms, a database is effectively a vector $x = (x_1, \ldots, x_l, \ldots, x_n)$ where x_l is the truth value of the lth ground atom ($x_l = 1$ if the atom appears in the database, and $x_l = 0$ otherwise). Given a database, MLN weights can in principle be learned using standard methods, as follows. If the ith formula has $n_i(x)$ true groundings in the data x, then by Equation 12.3 the derivative of the log-likelihood with respect to its weight is

$$\frac{\partial}{\partial w_i} \log P_w(X = x) = n_i(x) - \sum_{x'} P_w(X = x')\, n_i(x'), \tag{12.6}$$

where the sum is over all possible databases x', and $P_w(X = x')$ is $P(X = x')$ computed using the current weight vector $w = (w_1, \ldots, w_i, \ldots)$. In other words, the ith component of the gradient is simply the difference between the number of true groundings of the ith formula in the data and its expectation according to the current model. Unfortunately, counting the number of true groundings of a formula in a database is intractable, even when the formula is a single clause, as stated in the following proposition (due to Dan Suciu).

Proposition 12.4
Counting the number of true groundings of a first-order clause in a database is #P-complete in the length of the clause.

Proof Counting satisfying assignments of propositional monotone 2-CNF is #P-complete [49]. This problem can be reduced to counting the number of true groundings of a first-order clause in a database as follows. Consider a database composed of the ground atoms $R(0,1)$, $R(1,0)$, and $R(1,1)$. Given a monotone 2-CNF formula, construct a formula Φ that is a conjunction of predicates of the form $R(x_i, x_j)$, one for each disjunct $x_i \vee x_j$ appearing in the CNF formula. (For example, $(x_1 \vee x_2) \wedge (x_3 \vee x_4)$ would yield $R(x_1, x_2) \wedge R(x_3, x_4)$.) There is a one-to-one correspondence between the satisfying assignments of the 2-CNF and the true groundings of Φ. The latter are the false groundings of the clause formed by disjoining the negations of all the $R(x_i, x_j)$, and thus can be counted by counting the number of true groundings of this clause and subtracting it from the total number of groundings. ∎

In large domains, the number of true groundings of a formula may be counted approximately, by uniformly sampling groundings of the formula and checking whether they are true in the data. In smaller domains, and in our experiments below, we use an efficient recursive algorithm to find the exact count.

A second problem with (12.6) is that computing the expected number of true groundings is also intractable, requiring inference over the model. Further, efficient optimization methods also require computing the log-likelihood itself (12.3), and thus the partition function Z. This can be done approximately using a Monte Carlo maximum likelihood estimator (MC-MLE) [19]. However, in our experiments the Gibbs sampling used to compute the MC-MLEs and gradients did not converge in reasonable time, and using the samples from the unconverged chains yielded poor results.

A more efficient alternative, widely used in areas like spatial statistics, social network modeling, and language processing, is to optimize instead the pseudo-likelihood [3]

$$P_w^*(X{=}x) = \prod_{l=1}^{n} P_w(X_l{=}x_l | MB_x(X_l)), \tag{12.7}$$

where $MB_x(X_l)$ is the state of the Markov blanket of X_l in the data. The gradient of the pseudo-log-likelihood is

$$\frac{\partial}{\partial w_i} \log P^*_w(X{=}x) = \sum_{l=1}^{n} [n_i(x) - P_w(X_l{=}0|MB_x(X_l))\, n_i(x_{[X_l=0]}) \\ - P_w(X_l{=}1|MB_x(X_l))\, n_i(x_{[X_l=1]})], \qquad (12.8)$$

where $n_i(x_{[X_l=0]})$ is the number of true groundings of the ith formula when we force $X_l = 0$ and leave the remaining data unchanged, and similarly for $n_i(x_{[X_l=1]})$. Computing this expression (or (12.7)) does not require inference over the model. We optimize the pseudo-log-likelihood using the limited-memory BFGS algorithm [33]. The computation can be made more efficient in several ways:

- The sum in (12.8) can be greatly sped up by ignoring predicates that do not appear in the ith formula.
- The counts $n_i(x)$, $n_i(x_{[X_l=0]})$, and $n_i(x_{[X_l=1]})$ do not change with the weights, and need only be computed once (as opposed to in every iteration of BFGS).
- Ground formulas whose truth-value is unaffected by changing the truth-value of any single literal may be ignored, since then $n_i(x) = n_i(x_{[X_l=0]}) = n_i(x_{[X_l=1]})$. In particular, this holds for any clause which contains at least two true literals. This can often be the great majority of ground clauses.

To combat overfitting, we penalize the pseudo-likelihood with a Gaussian prior on each weight.

When we know a priori which predicates will be evidence, MLN weights can also be learned discriminatively [52].

ILP techniques can be used to learn additional clauses, refine the ones already in the MLN, or learn an MLN from scratch. Here we use the CLAUDIEN system for this purpose [10]. Unlike most other ILP systems, which learn only Horn clauses, CLAUDIEN is able to learn arbitrary first-order clauses, making it well suited to Markov logic. Also, by constructing a particular language bias, we are able to direct CLAUDIEN to search for refinements of the MLN structure. Alternatively, MLN structure can be learned by directly optimizing pseudo-likelihood [30].

12.9 Experiments

We have empirically tested the algorithms described in the previous sections using a database describing the Department of Computer Science and Engineering at the University of Washington (UW-CSE). The domain consists of 12 predicates and 2707 constants divided into 10 types. Types include: publication (342 constants), person (442), course (176), project (153), academic quarter (20), etc. Predicates include: `Professor(person)`, `Student(person)`, `Area(x, area)` (with `x` ranging over publications, persons, courses, and projects), `AuthorOf(publication, person)`, `AdvisedBy(person, person)`, `YearsInProgram(person, years)`, `CourseLevel(cour-se, level)`, `TaughtBy(course, person, quarter)`, `TeachingAssistant(course, person, quarter), etc. Additionally, there are 10 equality predicates: SamePerson (person, person), SameCourse(course, course), etc., which always have known, fixed values that are true iff the two arguments are the same constant.

Using typed variables, the total number of possible ground atoms (n in section 12.8) was 4,106,841. The database contained a total of 3380 tuples (i.e., there were 3380 true ground atoms). We obtained this database by scraping pages in the department's website (www.cs.washington.edu). Publications and AuthorOf relations were obtained by extracting from the BibServ database (www.bibserv.org) all records with author fields containing the names of at least two department members (in the form "last name, first name" or "last name, first initial").

We obtained a knowledge base by asking four volunteers to each provide a set of formulas in first-order logic describing the domain. (The volunteers were not shown the database of tuples, but were members of the department who thus had a general understanding about it.) Merging these yielded a KB of 96 formulas. The complete KB, volunteer instructions, database, and algorithm parameter settings are online at http://www.cs.washington.edu/ai/mln. Formulas in the KB include statements like: students are not professors; each student has at most one advisor; if a student is an author of a paper, so is her advisor; advanced students only TA courses taught by their advisors; at most one author of a given publication is a professor; students in phase I of the Ph.D. program have no advisor; etc. Notice that these statements are not always true, but are typically true.

For training and testing purposes, we divided the database into five subdatabases, one for each area: AI, graphics, programming languages, systems, and theory. Professors and courses were manually assigned to areas, and other constants were iteratively assigned to the most frequent area among other constants they appeared in some tuple with. Each tuple was then assigned to the area of the constants in it. Tuples involving constants of more than one area were discarded, to avoid train-test contamination. The subdatabases contained, on average, 521 true ground atoms out of a possible 58,457.

We performed leave-one-out testing by area, testing on each area in turn using the model trained from the remaining four. The test task was to predict the AdvisedBy(x, y) predicate given (a) all others (All Info) and (b) all others except Student(x) and Professor(x) (Partial Info). In both cases, we measured the average conditional log-likelihood of all possible groundings of AdvisedBy(x, y) over all areas, drew precision-recall curves, and computed the area under the curve. This task is an instance of link prediction, a problem that has been the object of much interest in statistical relational learning (see section 12.6). All KBs were converted to clausal form. Timing results are on a 2.8Ghz Pentium 4 machine.

12.9.1 Systems

In order to evaluate Markov logic, which uses logic and probability for inference, we wished to compare it with methods that use only logic or only probability. We were also interested in automatic induction of clauses using ILP techniques. This section gives details of the comparison systems used.

12.9.1.1 Logic

One important question we aimed to answer with the experiments is whether adding probability to a logical KB improves its ability to model the domain. Doing this requires observing the results of answering queries using only logical inference, but this is complicated by the fact that computing log-likelihood and the area under the precision-recall curve requires real-valued probabilities, or at least some measure of "confidence" in the truth of each ground atom being tested. We thus used the following approach. For a given knowledge base KB and set of evidence atoms E, let $\mathcal{X}_{KB \cup E}$ be the set of worlds that satisfy $KB \cup E$. The probability of a query atom q is then defined as $P(q) = \frac{|\mathcal{X}_{KB \cup E \cup q}|}{|\mathcal{X}_{KB \cup E}|}$, the fraction of $\mathcal{X}_{KB \cup E}$ in which q is true.

A more serious problem arises if the KB is inconsistent (which was indeed the case with the KB we collected from volunteers). In this case the denominator of $P(q)$ is zero. (Also, recall that an inconsistent KB trivally entails any arbitrary formula). To address this, we redefine $\mathcal{X}_{KB \cup E}$ to be the set of worlds which satisfies the maximum possible number of ground clauses. We use Gibbs sampling to sample from this set, with each chain initialized to a mode using WalkSat. At each Gibbs step, the step is taken with probability: 1 if the new state satisfies more clauses than the current one (since that means the current state should have 0 probability), 0.5 if the new state satisfies the same number of clauses (since the new and old state then have equal probability), and 0 if the new state satisfies fewer clauses. We then use only the states with the maximum number of satisfied clauses to compute probabilities. Notice that this is equivalent to using an MLN built from the KB and with all infinite equal weights.

12.9.1.2 Probability

The other question we wanted to answer with these experiments is whether existing (propositional) probabilistic models are already powerful enough to be used in relational domains without the need for the additional representational power provided by MLNs. In order to use such models, the domain must first be propositionalized by defining features that capture useful information about it. Creating good attributes for propositional learners in this highly relational domain is a difficult problem. Nevertheless, as a tradeoff between incorporating as much potentially relevant information as possible and avoiding extremely long feature vectors, we defined two sets of propositional attributes: order-1 and order-2. The former involves characteristics of individual constants in the query predicate, and the latter involves characteristics of relations between the constants in the query predicate.

For the order-1 attributes, we defined one variable for each (a, b) pair, where a is an argument of the query predicate and b is an argument of some predicate with the same value as a. The variable is the fraction of true groundings of this predicate in the data. Some examples of first-order attributes for $\mathtt{AdvisedBy(Matt, Pedro)}$ are: whether Pedro is a student, the fraction of publications that are published by Pedro, the fraction of courses for which Matt was a teaching assistant, etc.

The order-2 attributes were defined as follows: for a given (ground) query predicate $\mathtt{Q(q_1, q_2, \ldots, q_k)}$, consider all sets of k predicates and all assignments of constants $q_1, q_2, \ldots, q_k$ as arguments to the k predicates, with exactly one constant per predicate (in any order). For instance, if $\mathtt{Q}$ is $\mathtt{Advised - By(Matt, Pedro)}$ then one such possible set would be $\{\mathtt{TeachingAssistant(\_, Matt, \_), TaughtBy(\_, Pedro, \_)}\}$. This forms 2^k attributes of the example, each corresponding to a particular truth assignment to the k predicates. The value of an attribute is the number of times, in the training data the set of predicates have that particular truth assignment, when their unassigned arguments are all filled with the same constants. For example, consider filling the above empty arguments with "$\mathtt{CSE546}$" and "$\mathtt{Autumn\_0304}$." The resulting set, $\{\mathtt{TeachingAssistant(CSE546, Matt, Autumn\_0304)}$, $\mathtt{TaughtBy(CSE546, Pedro, Autumn\_0304)}\}$ has some truth assignment in the training data (e.g., {True,True}, {True,False}, . . .). One attribute is the number of such sets of constants that create the truth assignment {True,True}, another for {True,False}, and so on. Some examples of second-order attributes generated for the query $\mathtt{AdvisedBy(Matt, Pedro)}$ are: how often Matt is a teaching assistant for a course that Pedro taught (as well as how often he is not), how many publications Pedro and Matt have coauthored, etc.

The resulting 28 order-1 attributes and 120 order-2 attributes (for the All Info case) were discretized into five equal-frequency bins (based on the training set). We used two propositional learners: naive Bayes [14] and Bayesian networks [22] with structure and parameters learned using the VFBN2 algorithm [25] with a maximum of four parents per node. The order-2 attributes helped the naive Bayes classifier but hurt the performance of the Bayesian network classifier, so below we report results using the order-1 and order-2 attributes for naive Bayes, and only the order-1 attributes for Bayesian networks.

12.9.1.3 Inductive Logic Programming

Our original KB was acquired from volunteers, but we were also interested in whether it could have been developed automatically using ILP methods. As mentioned earlier, we used CLAUDIEN to induce a KB from data. CLAUDIEN was run with: local scope; minimum accuracy of 0.1; minimum coverage of 1; maximum complexity of 10; and breadth-first search. CLAUDIEN's search space is defined by its language bias. We constructed a language bias which allowed: a maximum of three variables in a clause; unlimited predicates in a clause; up to two non-negated appearances of a predicate in a clause, and two negated ones; and use of knowledge of predicate argument types. To minimize search, the equality predicates (e.g., SamePerson) were not used in CLAUDIEN, and this improved its results.

Besides inducing clauses from the training data, we were also interested in using data to automatically refine the KB provided by our volunteers. CLAUDIEN does not support this feature directly, but it can be emulated by an appropriately constructed language bias. We did this by, for each clause in the KB, allowing CLAUDIEN to (1) remove any number of the literals, (2) add up to v new variables, and (3) add up to l new literals. We ran CLAUDIEN for 24 hours on a Sun-Blade 1000 for each (v, l) in the set $\{(1, 2), (2, 3), (3, 4)\}$. All three gave nearly identical results; we report the results with $v = 3$ and $l = 4$.

12.9.1.4 Markov Logic

Our results compare the above systems to Markov logic. The MLNs were trained using a Gaussian weight prior with zero mean and unit variance, and with the weights initialized at the mode of the prior (zero). For optimization, we used the FORTRAN implementation of L-BFGS from Zhu et al. [58] and Byrd et al. [5], leaving all parameters at their default values, and with a convergence criterion (*ftol*) of 10^{-5}. Inference was performed using Gibbs sampling as described in section 12.7, with ten parallel Markov chains, each initialized to a mode of the distribution using MaxWalkSat. The number of Gibbs steps was determined using the criterion of DeGroot and Schervish [11][pp. 707 and 740-741]. Sampling continued until we reached a confidence of 95% that the probability estimate was within 1% of the true value in at least 95% of the nodes (ignoring nodes which are always true or false). A minimum of 1000 and maximum of 500,000 samples was used, with one sample per complete Gibbs pass through the variables. Typically, inference converged within 5000 to 100,000 passes. The results were insensitive to variation in the convergence thresholds.

12.9.2 Results

12.9.2.1 Training with MC-MLE

Our initial system used MC-MLE to train MLNs, with ten Gibbs chains, and each ground atom being initialized to true with the corresponding first-order predicate's probability of being true in the data. Gibbs steps may be taken quite quickly by noting that few counts of satisfied clauses will change on any given step. On the UW-CSE domain, our implementation took 4-5 ms per step. We used the maximum across all predicates of the Gelman criterion R [20] to determine when the chains had reached their stationary distribution. In order to speed convergence, our Gibbs sampler preferentially samples atoms that were true in either the data or the initial state of the chain. The intuition behind this is that most atoms are always false, and sampling repeatedly from them is inefficient. This improved convergence by approximately an order of magnitude over uniform selection of atoms. Despite these optimizations, the Gibbs sampler took a prohibitively long time to reach a reasonable convergence threshold (e.g., $R = 1.01$). After running for 24 hours (approximately 2 million Gibbs steps per chain), the average R-value across training sets was 3.04, with no one training set having reached an R-value less than 2 (other than briefly dipping to 1.5 in the early stages of the process). Considering this must be done iteratively as L-BFGS searches for the minimum, we estimate it would take anywhere from 20 to 400 days to complete the training, even with a weak convergence threshold such as $R = 2.0$. Experiments confirmed the poor quality of the models that resulted if we ignored the convergence threshold and limited the training process to less than ten hours. With a better choice of initial state, approximate counting, and improved MCMC techniques such as the Swendsen-Wang algorithm [15], MC-MLE may become practical, but it is not a viable option for training in the current version. (Notice that during learning MCMC is performed over the full ground network, which is too large to apply MaxWalkSat to.)

12.9.2.2 Training with Pseudo-likelihood

In contrast to MC-MLE, pseudo-likelihood training was quite fast. As discussed in section 12.8, each iteration of training may be done quite quickly once the initial clause and ground atom satisfiability counts are complete. On average (over the five test sets), finding these counts took 2.5 minutes. From there, training took, on average, 255 iterations of L-BFGS, for a total of 16 minutes.

12.9.2.3 Inference

Inference was also quite quick. Inferring the probability of all `AdvisedBy(x, y)` atoms in the All Info case took 3.3 minutes in the AI test set (4624 atoms), 24.4 in graphics (3721), 1.8 in programming languages (784), 10.4 in systems (5476), and 1.6 in theory (2704). The number of Gibbs passes ranged from 4270 to 500,000, and averaged 124,000. This amounts to 18 ms per Gibbs pass and approximately 200,000–500,000 Gibbs steps per second. The average time to perform inference in the Partial Info case was 14.8 minutes (vs. 8.3 in the All Info case).

12.9.2.4 Comparison of Systems

We compared twelve systems: the original KB (KB); CLAUDIEN (CL); CLAUDIEN with the original KB as language bias (CLB); the union of the original KB and CLAUDIEN's output in both cases (KB+CL and KB+CLB); an MLN with each of the above KBs (MLN(KB), MLN(CL), MLN(KB+CL), and MLN(KB+CLB)); naive Bayes (NB); and a Bayesian network learner (BN). Add-one smoothing of probabilities was used in all cases.

Table 12.4 summarizes the results, and figure 12.2 shows precision-recall curves for all areas (i.e., averaged over all `AdvisedBy(x, y)` predicates). MLNs are clearly more accurate than the alternatives, showing the promise of this approach. The purely logical and purely probabilistic methods often suffer when intermediate predicates have to be inferred, while MLNs are largely unaffected. Naive Bayes

Table 12.4 Experimental results for predicting `AdvisedBy(x, y)` when all other predicates are known (All Info) and when `Student(x)` and `Professor(x)` are unknown (Partial Info). CLL is the average conditional log-likelihood, and AUC is the area under the precision-recall curve. The results are averages over all atoms in the five test sets and their standard deviations. (See http://www.cs.washington.edu/ai/mln for details on how the standard deviations of the AUCs were computed.)

System	All Info		Partial Info	
	AUC	CLL	AUC	CLL
MLN(KB)	0.215 ± 0.0172	-0.052 ± 0.004	0.224 ± 0.0185	-0.048 ± 0.004
MLN(KB+CL)	0.152 ± 0.0165	-0.058 ± 0.005	0.203 ± 0.0196	-0.045 ± 0.004
MLN(KB+CLB)	0.011 ± 0.0003	-3.905 ± 0.048	0.011 ± 0.0003	-3.958 ± 0.048
MLN(CL)	0.035 ± 0.0008	-2.315 ± 0.030	0.032 ± 0.0009	-2.478 ± 0.030
MLN(CLB)	0.003 ± 0.0000	-0.052 ± 0.005	0.023 ± 0.0003	-0.338 ± 0.002
KB	0.059 ± 0.0081	-0.135 ± 0.005	0.048 ± 0.0058	-0.063 ± 0.004
KB+CL	0.037 ± 0.0012	-0.202 ± 0.008	0.028 ± 0.0012	-0.122 ± 0.006
KB+CLB	0.084 ± 0.0100	-0.056 ± 0.004	0.044 ± 0.0064	-0.051 ± 0.005
CL	0.048 ± 0.0009	-0.434 ± 0.012	0.037 ± 0.0001	-0.836 ± 0.017
CLB	0.003 ± 0.0000	-0.052 ± 0.005	0.010 ± 0.0001	-0.598 ± 0.003
NB	0.054 ± 0.0006	-1.214 ± 0.036	0.044 ± 0.0009	-1.140 ± 0.031
BN	0.015 ± 0.0006	-0.072 ± 0.003	0.015 ± 0.0007	-0.215 ± 0.003

performs well in AUC in some test sets, but very poorly in others; its CLLs are uniformly poor. CLAUDIEN performs poorly on its own, and produces no improvement when added to the KB in the MLN. Using CLAUDIEN to refine the KB typically performs worse in AUC but better in CLL than using CLAUDIEN from scratch; overall, the best-performing logical method is KB+CLB, but its results fall well short of the best MLNs. The general drop-off in precision at around 50% recall is attributable to the fact that the database is very incomplete, and only allows identifying a minority of the `AdvisedBy` relations. Inspection reveals that the occasional smaller drop-offs in precision at very low recalls are due to students who graduated or changed advisors after coauthoring many publications with them.

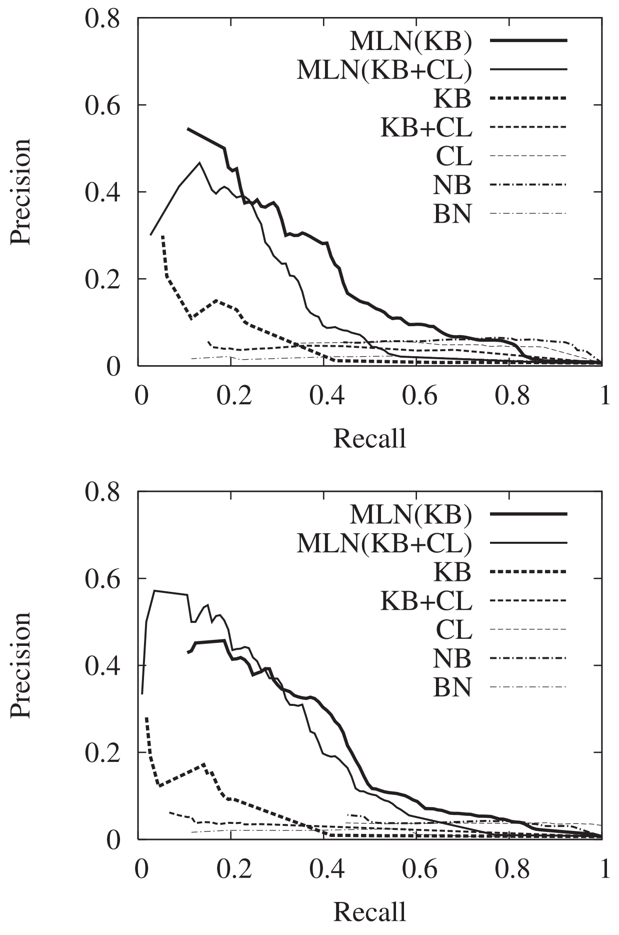

Figure 12.2 Precision and recall for all areas: All Info (upper graph) and Partial Info (lower graph).

12.10 Conclusion

The rapid growth in the variety of SRL approaches and tasks has led to the need for a unifying framework. In this chapter we propose *Markov logic* as a candidate for such a framework. Markov logic combines first-order logic and Markov networks and allows a wide variety of SRL tasks and approaches to be formulated in a common language. Initial experiments with an implementation of Markov logic have yielded good results. Software implementing Markov logic and learning and inference algorithms for it is available at http://www.cs.washington.edu/ai/alchemy.

Acknowledgments

We are grateful to Julian Besag, Vitor Santos Costa, James Cussens, Nilesh Dalvi, Alan Fern, Alon Halevy, Mark Handcock, Henry Kautz, Kristian Kersting, Tian Sang, Bart Selman, Dan Suciu, Jeremy Tantrum, and Wei Wei for helpful discussions. This research was partly supported by ONR grant N00014-02-1-0408 and by a Sloan Fellowship awarded to P. D. We used the VFML library in our experiments (http://www.cs.washington.edu/dm/vfml/).

References

[1] C. Anderson, P. Domingos, and D. Weld. Relational Markov models and their application to adaptive Web navigation. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 143–152, Edmonton, Canada, 2002. ACM Press.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

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

[4] W. Buntine. Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2:159–225, 1994.

[5] R. H. Byrd, P. Lu, and J. Nocedal. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing*, 16(5):1190–1208, 1995.

[6] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *Proceedings of ACM International Conference on Management of Data*, 1998.

[7] C. Cumby and D. Roth. Feature extraction languages for propositionalized relational learning. In *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*, 2003.

[8] J. Cussens. Individuals, relations and structures in probabilistic models. In *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*, 2003.

[9] J. Cussens. Loglinear models for first-order probabilistic reasoning. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1999.

[10] L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.

[11] M. H. DeGroot and M. J. Schervish. *Probability and Statistics, 3rd edition.* Addison Wesley, Boston, 2002.

[12] L. Dehaspe. Maximum entropy modeling with clausal constraints. In *Proceedings of the International Conference on Inductive Logic Programming*, 1997.

[13] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19: 380–392, 1997.

[14] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29:103–130, 1997.

[15] R.G. Edwards and A.G. Sokal. Generalization of the Fortuin-Kasteleyn-Swendsen-Wang representation and Monte Carlo algorithm. *Physics Review D*, 38:2009–2012, 1988.

[16] G. W. Flake, S. Lawrence, and C. L. Giles. Efficient identification of Web communities. In *International Conference on Knowledge Discovery and Data Mining*, 2000.

[17] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1999.

[18] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA, 1987.

[19] C. J. Geyer and E. A. Thompson. Constrained Monte Carlo maximum likelihood for dependent data. *Journal of the Royal Statistical Society, Series B*, 54(3):657–699, 1992.

[20] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, London, 1996.

[21] J. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1990.

[22] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[23] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, 2000.

[24] D. Heckerman, C. Meek, and D. Koller. Probabilistic entity-relationship models, PRMs, and plate models. In *Proceedings of the ICML-2004 Workshop on Statistical Relational Learning and Its Connections to Other Fields*, 2004.

[25] G. Hulten and P. Domingos. Mining complex models from arbitrarily large databases in constant time. In *International Conference on Knowledge Discovery and Data Mining*, 2002.

[26] M. Jaeger. On the complexity of inference about probabilistic relational models. *Artificial Intelligence*, 117:297–308, 2000.

[27] M. Jaeger. Reasoning about infinite random structures with relational Bayesian networks. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 1998.

[28] H. Kautz, B. Selman, and Y. Jiang. A general stochastic approach to solving problems with hard and soft constraints. In D. Gu, J. Du, and P. Pardalos, editors, *The Satisfiability Problem: Theory and Applications*, pages 573–586. American Mathematical Society, New York, 1997.

[29] K. Kersting and L. De Raedt. Towards combining inductive logic programming with Bayesian networks. In *Proceedings of the International Conference on Inductive Logic Programming*, 2001.

[30] S. Kok and P. Domingos. Learning the structure of Markov logic networks. In *Proceedings of the International Conference on Machine Learning*, 2005.

[31] J. Laffar and J.-L. Lassez. Constraint logic programming. In *Proceedings of the ACM Conference on Principles of Programming Languages*, 1987.

[32] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, Chichester, UK, 1994.

[33] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(3):503–528, 1989.

[34] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1987.

[35] E. Lloyd-Richardson, A. Kazura, C. Stanton, R. Niaura, and G. Papandonatos. Differentiating stages of smoking intensity among adolescents: Stage-specific psychological and social influences. *Journal of Consulting and Clinical Psychology*, 70(4), 2002.

[36] B. Milch, B. Marthi, and S. Russell. BLOG: Relational modeling with unknown objects. In *Proceedings of the ICML-2004 Workshop on Statistical Relational Learning and its Connections to Other Fields*, 2004.

[37] S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, Amsterdam, 1996.

[38] J. Neville and D. Jensen. Collective classification with relational dependency networks. In *Proceedings of the Second International Workshop on Multi-Relational Data Mining*, 2003.

[39] L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–177, 1997.

[40] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, New York, NY, 1999.

[41] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, 1988.

[42] D. Poole. First-order probabilistic inference. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2003.

[43] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.

[44] A. Popescul and L. H. Ungar. Structural logistic regression for link analysis. In *Proceedings of the Second International Workshop on Multi-Relational Data Mining*, 2003.

[45] A. Puech and S. Muggleton. A comparison of stochastic logic programs and Bayesian logic programs. In *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*, 2003.

[46] M. Richardson and P. Domingos. Building large knowledge bases by mass collaboration. In *Proceedings of the International Conference on Knowledge Capture*, 2003.

[47] S. Riezler. *Probabilistic Constraint Logic Programming.* PhD thesis, University of Tubingen, Tubingen, Germany, 1998.

[48] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[49] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82:273–302, 1996.

[50] V. Santos Costa, D. Page, M. Qazi, , and J. Cussens. CLP(BN): Constraint logic programming for probabilistic knowledge. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2003.

[51] T. Sato and Y. Kameya. PRISM: A symbolic-statistical modeling language. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1997.

[52] P. Singla and P. Domingos. Discriminative training of Markov logic networks. In *AAAI Press*, 2005.

[53] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2002.

[54] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications.* Cambridge University Press, Cambridge, UK, 1994.

[55] M. Wellman, J. S. Breese, and R. P. Goldman. From knowledge bases to decision models. *Knowledge Engineering Review*, 7:35–53, 1992.

[56] W. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, US Census Bureau, 1999.

[57] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. In *Proceedings of Neural Information Processing Systems*, 2001.

[58] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560, 1997.