Learning the Structure of Markov Logic Networks

Stanley Kok KOKS@CS.WASHINGTON.EDU
Pedro Domingos PEDROD@CS.WASHINGTON.EDU
Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA

Abstract

Markov logic networks (MLNs) combine logic and probability by attaching weights to first-order clauses, and viewing these as templates for features of Markov networks. In this paper we develop an algorithm for learning the structure of MLNs from relational databases, combining ideas from inductive logic programming (ILP) and feature induction in Markov networks. The algorithm performs a beam or shortest-first search of the space of clauses, guided by a weighted pseudo-likelihood measure. This requires computing the optimal weights for each candidate structure, but we show how this can be done efficiently. The algorithm can be used to learn an MLN from scratch, or to refine an existing knowledge base. We have applied it in two real-world domains, and found that it outperforms using off-the-shelf ILP systems to learn the MLN structure, as well as pure ILP, purely probabilistic and purely knowledge-based approaches.

1. Introduction

Statistical learning handles uncertainty in a robust and principled way. Relational learning (also known as inductive logic programming) models domains involving multiple relations. Recent years have seen a surge of interest in combining the two, driven by the realization that many (if not most) applications require both, and by the growing maturity of the two fields (Dietterich et al., 2003). Most approaches to date combine a logical language (e.g., Prolog, description logics) with Bayesian networks (e.g., Friedman et al. (1999); Kersting and De Raedt (2001)). However, the need to avoid cycles in Bayesian networks causes many difficulties when extending them to relational representations (Taskar et al., 2002). An alternative is to use undirected graphical models, also known as Markov networks or Markov random fields. This is the approach taken in relational Markov networks (Taskar et al., 2002). However, RMNs use a very restricted logical language (conjunctive database queries), and this limits the complexity of phenomena they can efficiently represent. In particular, RMNs require space exponential in the size of the cliques in the underlying Markov network. Recently, Richardson and Domingos (2004) introduced Markov logic networks (MLNs), which allow the features of the underlying Markov network to be specified by arbitrary formulas in finite first-order logic, and can compactly represent distributions involving large cliques.

Richardson and Domingos used an off-the-shelf ILP system (CLAUDIEN (De Raedt & Dehaspe, 1997)) to learn the structure of MLNs. This is unlikely to give the best results, because CLAUDIEN (like other ILP systems) is designed to simply find clauses that hold with some accuracy and frequency in the data, not to maximize the data's likelihood (and hence the quality of the MLN's probabilistic predictions). In this paper, we develop an algorithm for learning the structure of MLNs by directly optimizing a likelihood-type measure, and show experimentally that it outperforms the approach of Richardson and Domingos. The resulting system is arguably the most powerful combination to date of statistical and relational learning, while still having acceptable computational requirements.

We begin by briefly reviewing the necessary background in ILP (Section 2), Markov networks (Section 3), and Markov logic networks (Section 4). We then describe in detail our algorithm for structure learning in MLNs (Section 5). Finally, we report our experiments with the new algorithm and discuss their results (Section 6).

2. Logic and ILP

A *first-order knowledge base (KB)* is a set of sentences or formulas in first-order logic (Genesereth & Nilsson, 1987). Formulas 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.,

Appearing in *Proceedings of the 22^{nd} International Conference on Machine Learning*, Bonn, Germany, 2005. Copyright 2005 by the author(s)/owner(s).

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). 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 MotherOf(x) are terms. An *atomic formula* or *atom* is a predicate symbol applied to a tuple of terms (e.g., Friends(x, MotherOf(Anna))). 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. Formulas are recursively constructed from atomic formulas using logical connectives and quantifiers. A *positive literal* is an atomic formula; a *negative literal* is a negated atomic formula. A KB in *clausal form* is a conjunction of *clauses*, a clause being a disjunction of literals. A *definite clause* is a clause with exactly one positive literal (the *head*, with the negative literals constituting the *body*). A *possible world* or *Herbrand interpretation* assigns a truth value to each possible ground predicate.

ILP systems learn clausal KBs from relational databases, or refine existing KBs (Lavrač & Džeroski, 1994). In the *learning from entailment* setting, the system searches for clauses that entail all positive examples of some relation (e.g., Friends) and no negative ones. For example, FOIL (Quinlan, 1990) learns each definite clause by starting with the target relation as the head and greedily adding literals to the body. In the *learning from interpretations* setting, the examples are databases, and the system searches for clauses that are true in them. For example, CLAUDIEN (De Raedt & Dehaspe, 1997), starting with a trivially false clause, repeatedly forms all possible refinements of the current clauses by adding literals, and adds to the KB the ones that satisfy a minimum accuracy and coverage criterion.

MACCENT (Dehaspe, 1997) is an early example of a system that combines ILP with probability. It finds the maximum entropy distribution over a set of classes consistent with a set of constraints expressed as clauses. Like our algorithm, it builds on Markov network ideas; however, it only performs classification, while the goal here is to do general probability estimation (i.e., learn the joint distribution of all predicates). Also, MACCENT only allows deterministic background knowledge, while MLNs allow it to be uncertain; and MACCENT classifies each example separately, while MLNs allow for collective classification.

3. Markov Networks

A *Markov network* (also known as *Markov random field*) is a model for the joint distribution of a set of variables $X = (X_1, X_2, \ldots, X_n) \in \mathcal{X}$ (Della Pietra et al., 1997). 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

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

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

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_j w_j f_j(x)\right) \tag{2}$$

A feature may be any real-valued function of the state. This paper will focus on binary features, $f_j(x) \in \{0, 1\}$. In the most direct translation from the potential-function form (Equation 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. MLNs take advantage of this.

Markov network weights have traditionally been learned using iterative scaling (Della Pietra et al., 1997). However, maximizing the likelihood (or posterior) using a quasi-Newton optimization method like L-BFGS has recently been found to be much faster (Sha & Pereira, 2003). Work on learning the structure (i.e., the features) of Markov networks has been relatively sparse to date. Della Pietra et al. (1997) induce conjunctive features by starting with a set of atomic features (the original variables), conjoining each current feature with each atomic feature, adding to the network the conjunction that most increases likelihood, and repeating. McCallum (2003) extends this to the case of conditional random fields, which are Markov networks trained to maximize the conditional likelihood of a set of outputs given a set of inputs.

4. Markov Logic Networks

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 MLNs is to soften these constraints: when a world violates one formula in the KB it is less probable, but not impossible. The fewer formulas 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.

Definition 4.1 *(Richardson & Domingos, 2004) A Markov logic network 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 $M_{L,C}$ (Equations 1 and 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 predicate 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.*

Thus there is an edge between two nodes of $M_{L,C}$ iff the corresponding ground predicates appear together in at least one grounding of one formula in L. An MLN can be viewed as a *template* for constructing Markov networks. From Definition 4.1 and Equations 1 and 2, the probability distribution over possible worlds x specified by the ground Markov network $M_{L,C}$ is given by

$$P(X=x) = \frac{1}{Z} \exp\left(\sum_{i=1}^{F} w_i n_i(x)\right) \quad (3)$$

where F is the number formulas in the MLN and $n_i(x)$ is the number of true groundings of F_i in x. As formula weights increase, an MLN increasingly resembles a purely logical KB, becoming equivalent to one in the limit of all infinite weights.

In this paper we will focus on MLNs whose formulas are function-free clauses and assume domain closure, ensuring that the Markov networks generated are finite (Richardson & Domingos, 2004). In this case, the groundings of a formula are formed simply by replacing its variables with constants in all possible ways. For example, if $C = \{\texttt{Anna}, \texttt{Bob}\}$, the formula $\forall \texttt{x}\ \texttt{Smokes(x)} \Rightarrow \texttt{Cancer(x)}$ in the MLN L yields the features $\texttt{Smokes(Anna)} \Rightarrow \texttt{Cancer(Anna)}$ and $\texttt{Smokes(Bob)} \Rightarrow \texttt{Cancer(Bob)}$ in the ground Markov network $M_{L,C}$. See Richardson and Domingos (2004, Table II) for details.

MLN weights can be learned by maximizing the likelihood of a relational database. (As often in ILP, a closed-world assumption is made, whereby all ground atoms not in the database are assumed false.) However, as in Markov networks, this requires computing the expected number of true groundings of each formula, which can take exponential time. Although this computation can be done approximately using Markov chain Monte Carlo inference (Della Pietra et al., 1997), Richardson and Domingos found this to be too slow. Instead, they maximized the pseudo-likelihood of the data, a widely-used alternative (Besag, 1975). If x is a possible world (relational database) and x_l is the lth ground atom's truth value, the pseudo-log-likelihood of x given weights w is

$$\log P^*_w(X=x) = \sum_{l=1}^{n} \log P_w(X_l = x_l | MB_x(X_l)) \quad (4)$$

where $MB_x(X_l)$ is the state of X_l's *Markov blanket* in the data (i.e., the truth values of the ground atoms it appears in some ground formula with), and

$$P(X_l = x_l | MB_x(X_l)) = \frac{\exp\left(\sum_{i=1}^{F} w_i n_i(x)\right)}{\exp\left(\sum_{i=1}^{F} w_i n_i(x_{[X_l=0]})\right) + \exp\left(\sum_{i=1}^{F} w_i n_i(x_{[X_l=1]})\right)} \quad (5)$$

where $n_i(x)$ is the number of true groundings of the ith formula in x, $n_i(x_{[X_l=0]})$ 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 $n_i(x_{[X_l=1]})$.

Singla and Domingos (2005) proposed a discriminative approach to learning MLN weights, but did not learn MLN structure, like we do here. Richardson and Domingos first learned the structure of an MLN using CLAUDIEN, and then learned maximum pseudo-likelihood weights for it using L-BFGS. In the next section we propose a sounder approach.

5. Structure Learning in MLNs

MLN structure learning can start from an empty network or from an existing KB. Either way, like Richardson and Domingos (2004), we have found it useful to start by adding all unit clauses (single predicates) to the MLN. The weights of these capture (roughly speaking) the marginal distributions of the predicates, allowing the longer clauses to focus on modeling predicate dependencies.

The design space for MLN structure learning algorithms includes the choice of evaluation measure, clause construction operators, search strategy, and speedup methods. We discuss each of these in turn.

5.1. Evaluation Measures

We initially used the same pseudo-likelihood measure as Richardson and Domingos (Equation 4). However, we found this to give undue weight to the largest-arity predicates, resulting in poor modeling of the rest. We thus defined the weighted pseudo-log-likelihood (WPLL) as

$$\log P^\bullet_w(X=x) = \sum_{r \in R} c_r \sum_{k=1}^{g_r} \log P_w(X_{r,k} = x_{r,k} | MB_x(X_{r,k})) \quad (6)$$

where R is the set of first-order predicates, g_r is the number of groundings of first-order predicate r, and $x_{r,k}$ is the truth value (0 or 1) of the kth grounding of r. The choice of predicate weights c_r depends on the user's goals. In our experiments, we simply set $c_r = 1/g_r$, which has the effect of weighting all first-order predicates equally. If modeling a predicate is not important (e.g., because it will always be part of the evidence), we set its weight to zero. We used WPLL in all versions of MLN learning in our experiments. To combat overfitting, we penalize the WPLL with a structure prior of $e^{-\alpha \sum_{i=1}^{F} d_i}$, where d_i is the number of predicates that differ between the current version of the clause and the original one. (If the clause is new, this is simply its length.) This is similar to the approach used in learning Bayesian networks (Heckerman et al., 1995). Following Richardson and Domingos, we also penalize each weight with a Gaussian prior.

A potentially serious problem that arises when evaluating candidate clauses using WPLL is that the optimal (maximum WPLL) weights need to be computed for each candidate. Given that this involves numerical optimization, and may need to be done thousands or millions of times, it could easily make the algorithm too slow to be practical. Indeed, in the UW-CSE domain (see next section), we found that learning the weights using L-BFGS took 3 minutes on average, which is fast enough if only done once, but unfeasible to do for every candidate clause. Della Pietra et al. (1997) and McCallum (2003) address this problem by assuming that the weights of previous features do not change when testing a new one. Surprisingly, we found this to be unnecessary if we use the very simple approach of initializing L-BFGS with the current weights (and zero weight for a new clause). Although in principle all weights could change as the result of introducing or modifying a clause, in practice this seldom happens. Second-order, quadratic-convergence methods like L-BFGS are known to be very fast if started near the optimum. This is what happens in our case; L-BFGS typically converges in just a few iterations, sometimes one. The time required to evaluate a clause is in fact dominated by the time required to compute the number of its true groundings in the data, and this is a problem we focus on in Subsection 5.4.

5.2. Clause Construction Operators

When learning an MLN from scratch (i.e., from a set of unit clauses), the natural operator to use is the addition of a literal to a clause. When refining a hand-coded KB, the goal is to correct the errors made by the human experts. These errors include omitting conditions from rules and including spurious ones, and can be corrected by operators that add and remove literals from a clause. These are the basic operators that we use. In addition, we have found that many common errors (wrong direction of implication, wrong use of connectives with quantifiers, etc.) can be corrected at the clause level by flipping the signs of predicates, and we also allow this. When adding a literal to a clause, we consider all possible ways in which the literal's variables can be shared with existing ones, subject to the constraint that the new literal must contain at least one variable that appears in an existing one. To control the size of the search space, we set a limit on the number of distinct variables in a clause. We only try removing literals from the original hand-coded clauses or their descendants, and we only consider removing a literal if it leaves at least one path of shared variables between each pair of remaining literals.

5.3. Search Strategies

We have implemented two search strategies, one faster and one more complete. The first approach adds clauses to the MLN one at a time, using beam search to find the best clause to add: starting with the unit clauses and the expert-supplied ones, we apply each legal literal addition and deletion to each clause, keep the b best ones, apply the operators to those, and repeat until no new clause improves the WPLL. The chosen clause is the one with highest WPLL found in any iteration of the search. If the new clause is a refinement of a hand-coded one, it replaces it. (Notice that, even though we both add and delete literals, no loops can occur because each change must improve WPLL to be accepted.)

The second approach adds k clauses at a time to the MLN, and is similar to that of McCallum (2003). In contrast to beam search, which adds the best clause of any length found, this approach adds all "good" clauses of length l before attempting any of length $l + 1$. We call it *shortest-first search*.

Table 1 shows the structure learning algorithm in pseudocode, Table 2 shows beam search, and Table 3 shows shortest-first search for the case where the initial MLN contains only unit clauses.

5.4. Speedup Techniques

The algorithms described in the previous section may be very slow, particularly in large domains. However, they can be greatly sped up using a combination of techniques that we now describe.

- Richardson and Domingos (2004) list several ways of speeding up the computation of the pseudo-log-likelihood and its gradient, and we apply them to the WPLL (Equation 6). In addition, in Equation 5 we ignore all clauses that the predicate does not appear in.
- When learning MLN weights to evaluate candidate

Table 1. MLN structure learning algorithm.

```
function StructLearn($R$, $MLN$, $DB$)
  inputs: $R$, a set of predicates
          $MLN$, a clausal Markov logic network
          $DB$, a relational database
  output: Modified $MLN$
  Add all unit clauses from $R$ to $MLN$
  for each non-unit clause $c$ in $MLN$ (optionally)
    Try all combinations of sign flips of literals in $c$, and
    keep the one that gives the highest WPLL($MLN$, $DB$)
  $Clauses_0 \leftarrow$ {All clauses in $MLN$}
  LearnWeights($MLN$, $DB$)
  $Score \leftarrow$ WPLL($MLN$, $DB$)
  repeat
    $Clauses \leftarrow$ FindBestClauses($R$, $MLN$, $Score$, $Clauses_0$, $DB$)
    if $Clauses \neq \emptyset$
      Add $Clauses$ to $MLN$
      LearnWeights($MLN$, $DB$)
      $Score \leftarrow$ WPLL($MLN$, $DB$)
  until $Clauses = \emptyset$
  for each non-unit clause $c$ in $MLN$
    Prune $c$ from $MLN$ unless this decreases WPLL($MLN$, $DB$)
  return $MLN$
```

Table 2. Beam search for the best clause.

```
function FindBestClauses($R$, $MLN$, $Score$, $Clauses_0$, $DB$)
  inputs: $R$, a set of predicates
          $MLN$, a clausal Markov logic network
          $Score$, WPLL of $MLN$
          $Clauses_0$, a set of clauses
          $DB$, a relational database
  output: $BestClause$, a clause to be added to $MLN$
  $BestClause \leftarrow \emptyset$
  $BestGain \leftarrow 0$
  $Beam \leftarrow Clauses_0$
  Save the weights of the clauses in $MLN$
  repeat
    $Candidates \leftarrow$ CreateCandidateClauses($Beam$, $R$)
    for each clause $c \in Candidates$
      Add $c$ to $MLN$
      LearnWeights($MLN$, $DB$)
      $Gain(c) \leftarrow$ WPLL($MLN$, $DB$) $- Score$
      Remove $c$ from $MLN$
      Restore the weights of the clauses in $MLN$
    $Beam \leftarrow$ {The $b$ clauses $c \in Candidates$ with highest
            $Gain(c) > 0$ and with $Weight(c) > \epsilon > 0$ }
    if $Gain$(Best clause $c^*$ in $Beam$) $> BestGain$
      $BestClause \leftarrow c^*$
      $BestGain \leftarrow Gain(c^*)$
  until $Beam = \emptyset$ or $BestGain$ has not changed in two iterations
  return $\{BestClause\}$
```

clauses, we use a looser convergence threshold and lower maximum number of iterations for L-BFGS than when updating the MLN with the chosen clause(s).

- We compute the contribution of a predicate to the WPLL approximately by uniformly sampling a fraction of its groundings (true and false ones separately), computing the conditional likelihood of each one (Equation 5), and extrapolating the average. The number of samples can be chosen to guarantee that, with high confidence, the chosen clause(s) are the same that would be obtained if we computed the WPLL exactly. This effectively makes the runtime of the WPLL computation independent of the number of predicate groundings (Hulten & Domingos, 2002). At the end of the algorithm we do a final round of weight learning without subsampling.

- We use a similar strategy to compute the number of true groundings of a clause, required for the WPLL and its gradient. In particular, we use the algorithm of Karp and Luby (1983). In practice, we found that the estimates converge much faster than the algorithm specifies, so we run the convergence test of DeGroot and Schervish (2002, p. 707) after every 100 samples and terminate if it succeeds. In addition, we use looser convergence criteria during candidate clause evaluation than during update with the chosen clause. (See Sebag and Rouveirol (1997) for a related use of sampling in ILP.)

- When most clause weights do not change significantly with each run of L-BFGS, neither do most conditional log-likelihoods (CLLs) of ground predicates (log of Equation 5). We take advantage of this by storing the CLL of each sampled ground predicate, and only recomputing it if a clause weight affecting it changes by more than some threshold δ. When a CLL changes, we subtract its old value from the total WPLL and add the new one. The computation of the gradient of the WPLL is similarly optimized.

- We use a lexicographic ordering on clauses to avoid redundant computations for clauses that are syntactically identical. (However, we do not detect clauses that are semantically equivalent but syntactically different; this is an NP-complete problem.) We also cache the new clauses created during each search and their counts, avoiding the need to recompute them in later searches.

6. Experiments

6.1. Databases

We carried out experiments on two publicly-available databases: the UW-CSE database used by Richardson and Domingos (2004) (available at http://www.cs.washington.edu/ai/mln), and McCallum's Cora database of computer science citations as segmented by Bilenko and Mooney (2003) (available at http://www.cs.utexas.edu/users/ml/riddle/data/cora.tar.gz).

The UW-CSE domain consists of 22 predicates and 1158 constants divided into 10 types. Types include: publication, person, course, etc. Predicates include: `Professor(person)`, `AdvisedBy(person1, person2)`, etc. Using typed variables, the total number of possible ground predicates is 4,055,575. The database contains a total of 3212 tuples (ground atoms). We used the hand-coded knowledge base provided with it, which includes 94

Table 3. Shortest-first search for the best clauses.

```
function FindBestClauses($R$, $MLN$, $Score$, $Clauses_0$, $DB$)
  inputs: $R$, a set of predicates
          $MLN$, a clausal Markov logic network
          $Score$, WPLL of $MLN$
          $Clauses_0$, a set of clauses
          $DB$, a relational database
  output: $BestClauses$, a set of clauses to be added to $MLN$
  Save the weights of the clauses in $MLN$
  if this is the first time FindBestClauses is called
    $Candidates \leftarrow \emptyset$
    $l \leftarrow 1$
  repeat
    if $l = 1$ or this is not the first iteration of the repeat loop
      if there is no clause in $Candidates$ of length $< l$
        that was not previously extended
        $l \leftarrow l + 1$
      $Clauses \leftarrow$ {Clauses of length $l-1$ in $MLN$ not previ-
        ously extended } $\cup$ {$s$ best clauses of length $l-1$ in
        $Candidates$ not previously extended }
      $Candidates \leftarrow Candidates \cup$
        CreateCandidateClauses($Clauses$, $R$)
    for each clause $c \in Candidates$ not previously evaluated
      Add $c$ to $MLN$
      LearnWeights($MLN$, $DB$)
      $Gain(c) \leftarrow$ WPLL($MLN$, $DB$) $- Score$
      Remove $c$ from $MLN$
      Restore the weights of the clauses in $MLN$
    $Candidates \leftarrow \{m$ best clauses in $Candidates\}$
  until $l = l_{max}$ or there is a clause $c \in Candidates$ with
    $Gain(c) > 0$ and $Weight(c) > \epsilon > 0$
  $BestClauses \leftarrow$ {The $k$ clauses $c \in Candidates$ with highest
              $Gain(c) > 0$ and with $Weight(c) > \epsilon > 0$ }
  $Candidates \leftarrow Candidates \setminus BestClauses$
  return $BestClauses$
```

formulas stating regularities like: each student has at most one advisor; if a student is an author of a paper, so is her advisor; etc. Notice that these statements are not always true, but are typically true.

The Cora dataset is a collection of 1295 different citations to 112 computer science research papers. We used the author, venue, title and year fields. The goal is to determine which pairs of citations refer to the same paper (i.e., to infer the truth values of all groundings of `SameCitation(c1, c2)`). These values are available in the data. Additionally, we can attempt to deduplicate the author, title and venue strings, and we labeled these manually. We defined predicates for each field that discretize the percentage of words that two strings have in common. For example, `WordsInCommonInTitle20%(title1, title2)` is true iff the two titles have 0-20% of their words in common. These predicates are always given as evidence, and we do not attempt to predict them. Using typed variables, the total number of possible ground predicates is 5,225,411. The database contained a total of 378,589 tuples (ground atoms). A hand-crafted KB for this domain was provided by a colleague; it contains 26 clauses stating regularities like: if two citations are the same, their authors, venues, etc., are the same, and vice-versa; if two fields of the same type have many words in common, they are the same; etc.

6.2. Systems

We compared nine versions of MLN learning: weight learning applied to the hand-coded KB (MLN(KB)); structure learning using CLAUDIEN, FOIL and Aleph (Srinivasan, 2000) followed by weight learning (respectively MLN(CL), MLN(FO) and MLN(AL)); structure learning using CLAUDIEN with the KB providing the language bias as in Richardson and Domingos (2004), followed by weight learning on the output of CLAUDIEN merged with the KB (MLN(KB+CL)); structure learning using our algorithm with beam search, starting from an empty MLN (MLN(SLB)), starting from the hand-coded KB (MLN(KB+SLB)), and starting from an empty MLN but allowing hand-coded clauses to be added during the first search step (MLN(SLB+KB)); and structure learning using our algorithm with shortest-first search, starting from an empty MLN (MLN(SLS)). We added unit clauses to all nine systems. In addition, we compared MLN learning with three pure ILP approaches (CLAUDIEN (CL), FOIL (FO), and Aleph (AL)), a pure knowledge-based approach (the hand-coded KB (KB)), the combination of CLAUDIEN and the hand-coded KB as described above (KB+CL), and two pure probabilistic approaches (naive Bayes (NB) (Domingos & Pazzani, 1997) and Bayesian networks (BN) (Heckerman et al., 1995)). Notice that ILP learners like FOIL and Aleph are not directly comparable with MLNs (or CLAUDIEN), because they only learn to predict designated target predicates, as opposed to finding arbitrary regularities over all predicates. For an approximate comparison, we used FOIL and Aleph to learn rules with each predicate as the target in turn.

We used the algorithm of Richardson and Domingos to construct attributes for the naive Bayes and Bayesian network learners. Since our goal is to measure predictive performance over all predicates, not just the `AdvisedBy(x, y)` predicate that Domingos and Richardson focused on, we learned a naive Bayes classifier and a Bayesian network for each predicate. (Attempting to learn a single Bayesian network to predict all predicates simultaneously gave very poor results; this is not surprising, since in this case there are only four examples to learn from – one per training area.) Following Richardson and Domingos, we tried using order-1 and order-1+2 attributes, and report the best results.

We used the same settings for CLAUDIEN as Richardson and Domingos, and let CLAUDIEN run for 24 hours on a Sun Blade 1000 workstation.[1] We used the default FOIL parameter settings except for the maximum number of variables per clause, which we set to 5 (UW-CSE) and 6 (Cora), and the minimum clause accuracy, which we set to 50%. For Aleph, we used all of the default settings except for the maximum clause length, which we set to 4 (UW-CSE) and

[1]CLAUDIEN only runs on Solaris machines.

7 (Cora). The parameters used for our structure learning algorithms were as follows: $\alpha = 0.01$ (UW-CSE) and 0.001 (Cora); maximum variables per clause = 5 (UW-CSE) and 6 (Cora);[2] $\epsilon = 1$ (UW-CSE) and 0.01 (Cora); $\delta = 10^{-4}$; $s = 200$; $m = 100,000$; $l_{max} = 3$ (UW-CSE) and 7 (Cora); and $k = 10$ (UW-CSE) and 1 (Cora). L-BFGS was run with the following parameters: maximum iterations = 10,000 (tight) and 10 (loose); convergence threshold $= 10^{-5}$ (tight) and 10^{-4} (loose). The mean and variance of the Gaussian prior were set to 0 and 100, respectively, in all runs. Parameters were set in an *ad hoc* manner, and per-fold optimization using a validation set could conceivably yield better results.

6.3. Methodology

In the UW-CSE domain, we used the same leave-one-area-out methodology as Richardson and Domingos (2004). In the Cora domain, we performed five runs with train-test splits of approximately equal size, ensuring that no true set of matching records was split between train and test sets to avoid contamination. For each system on each test set, we measured the conditional log-likelihood (CLL) and area under the precision-recall curve (AUC) for each predicate. The advantage of the CLL is that it directly measures the quality of the probability estimates produced. The advantage of the AUC is that it is insensitive to the large number of true negatives (i.e., ground atoms that are false and predicted to be false). The CLL of a predicate is the average over all its groundings of the log of Equation 5, which we smoothed by using a weighted average of this value with a prior of 0.5 (with weights of 0.99 and 0.01, respectively). The precision-recall curve for a predicate is computed by varying the threshold CLL above which a ground atom is predicted to be true. For both CLL and AUC, the values we report are averages over all predicates (in the UW-CSE domain) or all non-evidence predicates (in the Cora domain), with all predicates weighted equally. We computed the standard deviations of the AUCs using the method of Richardson and Domingos (2004). To obtain probabilities from the ILP models and hand-coded KBs (required to compute CLLs and AUCs), we treated them as MLNs with all equal infinite weights.

6.4. Results

The results on the UW-CSE domain are shown in Table 4, and the results on Cora are shown in Table 5.[3] All versions of our MLN structure learning algorithm greatly outperform using ILP systems to learn MLN structure, in both CLL and AUC, in both domains. This is consistent with our hypothesis that directly optimizing (pseudo-)likelihood when learning structure yields better models. In both domains, shortest-first search starting from an empty network (MLN(SLS)) gives the best overall results, but is much slower than beam search (MLN(SLB)) (see below).

The purely logical approaches (CL, FO, AL, KB and KB+CL) did quite poorly. This occurred because they assigned very low probabilities to true ground atoms whenever they were not entailed by the logical KB, and this occurred quite often. Learning weights for the hand-coded KBs was quite helpful, confirming the utility of transforming KBs into MLNs. However, structure learning gave the best overall results. In the UW-CSE domain, refining the hand-coded KB (MLN(KB+SLB)) did not improve

Table 4. Experimental results on the UW-CSE database.

System	CLL	AUC
MLN(SLS)	-0.061 ± 0.004	0.533 ± 0.003
MLN(SLB)	-0.088 ± 0.005	0.472 ± 0.004
MLN(KB+SLB)	-0.140 ± 0.005	0.430 ± 0.003
MLN(SLB+KB)	-0.071 ± 0.005	0.551 ± 0.003
MLN(KB+CL)	-0.115 ± 0.005	0.506 ± 0.004
MLN(CL)	-0.151 ± 0.005	0.306 ± 0.001
MLN(FO)	-0.208 ± 0.006	0.140 ± 0.000
MLN(AL)	-0.223 ± 0.006	0.148 ± 0.001
MLN(KB)	-0.142 ± 0.005	0.429 ± 0.003
KB+CL	-0.789 ± 0.012	0.318 ± 0.003
CL	-0.574 ± 0.010	0.170 ± 0.004
FO	-0.661 ± 0.003	0.131 ± 0.001
AL	-0.579 ± 0.006	0.117 ± 0.000
KB	-0.812 ± 0.011	0.266 ± 0.003
NB	-0.370 ± 0.005	0.390 ± 0.003
BN	-0.166 ± 0.004	0.397 ± 0.002

Table 5. Experimental results on the Cora database.

System	CLL	AUC
MLN(SLS)	-0.054 ± 0.000	0.813 ± 0.001
MLN(SLB)	-0.058 ± 0.000	0.782 ± 0.001
MLN(KB+SLB)	-0.055 ± 0.000	0.828 ± 0.001
MLN(SLB+KB)	-0.058 ± 0.000	0.782 ± 0.001
MLN(KB+CL)	-0.069 ± 0.000	0.799 ± 0.001
MLN(CL)	-0.158 ± 0.001	0.148 ± 0.000
MLN(FO)	-0.213 ± 0.000	0.529 ± 0.001
MLN(KB)	-0.066 ± 0.000	0.809 ± 0.001
KB+CL	-0.191 ± 0.001	0.658 ± 0.001
CL	-0.693 ± 0.000	0.148 ± 0.000
FO	-0.717 ± 0.001	0.583 ± 0.001
KB	-0.229 ± 0.001	0.657 ± 0.001
NB	-0.411 ± 0.001	0.096 ± 0.001
BN	-0.257 ± 0.001	0.107 ± 0.000

[2] In the Cora domain, we further sped up learning by using syntactic restrictions on clauses similar to CLAUDIEN's declarative bias; details are in an online appendix at http://www.cs.washington.edu/ai/lsmln.

[3] We tried Aleph with many different parameter settings on Cora, but it always crashed by running out of memory.

on learning from scratch. Structure learning was unable to break out of the local optimum represented by MLN(KB), leading to poor performance. This problem is overcome if we start instead from an empty KB but allow hand-coded clauses to be added during the first step of beam search (MLN(SLB+KB)).

MLNs also greatly outperformed the purely probabilistic approaches (NB and BN). This was particularly noticeable in the UW-CSE domain, which contains little conventional attribute-value data but much relational information.

In the UW-CSE domain, shortest-first search without the speedups described in Subsection 5.4 did not finish running in 24 hours on a cluster of 15 dual-CPU 2.8 GHz Pentium 4 machines. With the speedups, it took an average of 5.3 hours. For beam search, the speedups reduced average running time from 13.7 hours to 8.8 hours on a standard 2.8 GHz Pentium 4 CPU. To investigate the contribution of each speed-up technique, we reran shortest-first search on one fold of the UW-CSE domain, leaving out one technique at a time. Clause and predicate sampling provide the largest speedups (six-fold and five-fold, respectively), and weight thresholding the smallest (1.025). None of the techniques adversely affect AUC, and predicate sampling is the only one that significantly reduces CLL (disabling it improves CLL from -0.059 to -0.038). Inference times were relatively short, taking a maximum of 1 minute (UW-CSE) and 12 minutes (Cora) on a standard 2.8 GHz Pentium 4 CPU.

We have also applied our algorithms to two time-changing domains, and shown that they outperform pure ILP and purely probabilistic approaches there (Sanghai et al., 2005).

In summary, both our algorithms are effective; we recommend using shortest-first search when accuracy is paramount, and beam search when speed is a concern.

7. Conclusion and Future Work

Markov logic networks are a powerful combination of first-order logic and probability. We have introduced an algorithm to automatically learn first-order clauses and their weights in MLNs. Empirical tests with real-world data in two domains have shown the promise of our approach.

Directions for future work include speeding up the counting of the number of true groundings of a first-order clause (the most significant bottleneck in our algorithm), studying alternate weight learning approaches, probabilistically bounding the loss in accuracy due to subsampling, and extending our approach to probabilistic predicate discovery.

Acknowledgments: We thank Matt Richardson and Parag Singla for helpful discussions. This work was partly funded by ONR grant N00014-02-1-0408, and by a Sloan Fellowship and an NSF CAREER Award to the second author.

References

Besag, J. (1975). Statistical analysis of non-lattice data. *The Statistician*, *24*, 179–195.

Bilenko, M., & Mooney, R. (2003). Adaptive duplicate detection using learnable string similarity measures. *Proc. KDD-03* (pp. 39–48).

De Raedt, L., & Dehaspe, L. (1997). Clausal discovery. *Machine Learning*, *26*, 99–146.

DeGroot, M. H., & Schervish, M. J. (2002). *Probability and statistics*. Boston, MA: Addison Wesley. 3rd edition.

Dehaspe, L. (1997). Maximum entropy modeling with clausal constraints. *Proc. ILP-97* (pp. 109–125).

Della Pietra, S., Della Pietra, V., & Lafferty, J. (1997). Inducing features of random fields. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, *19*, 380–392.

Dietterich, T., Getoor, L., & Murphy, K. (Eds.). (2003). *Proc. ICML-2004 Wkshp. on Statistical Relational Learning*. Banff, Canada.

Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, *29*, 103–130.

Friedman, N., Getoor, L., Koller, D., & Pfeffer, A. (1999). Learning probabilistic relational models. *Proc. IJCAI-99* (pp. 1300–1307).

Genesereth, M. R., & Nilsson, N. J. (1987). *Logical foundations of artificial intelligence*. San Mateo, CA: Morgan Kaufmann.

Heckerman, D., Geiger, D., & Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge & statistical data. *Machine Learning*, *20*, 197–243.

Hulten, G., & Domingos, P. (2002). Mining complex models from arbitrarily large databases in constant time. *Proc. KDD-02* (pp. 525–531).

Karp, R., & Luby, M. (1983). Monte Carlo algorithms for enumeration and reliability problems. *Proc. FOCS-83* (pp. 56–64).

Kersting, K., & De Raedt, L. (2001). Towards combining inductive logic programming with Bayesian networks. *Proc. ILP-01* (pp. 118–131).

Lavrač, N., & Džeroski, S. (1994). *Inductive logic programming: Techniques and applications*. Chichester, UK: Ellis Horwood.

McCallum, A. (2003). Efficiently inducing features of conditional random fields. *Proc. UAI-03* (pp. 403–410).

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*, 239–266.

Richardson, M., & Domingos, P. (2004). *Markov logic networks* (Tech. Rept.). Dept. Comp. Sci. & Eng., Univ. Washington, Seattle. http://www.cs.washington.edu/homes/pedrod/mln.pdf.

Sanghai, S., Domingos, P., & Weld, D. (2005). *Learning models of relational stochastic processes* (Tech. Rept.). Dept. Comp. Sci. & Eng., Univ. Washington, Seattle. http://www.-cs.washington.edu/homes/pedrod/lmrsp.pdf.

Sebag, M., & Rouveirol, C. (1997). Tractable induction and classification in first-order logic via stochastic matching. *Proc. IJCAI-97* (pp. 888–893).

Sha, F., & Pereira, F. (2003). Shallow parsing with conditional random fields. *Proc. HLT-NAACL-03* (pp. 134–141).

Singla, P., & Domingos, P. (2005). Discriminative training of Markov logic networks. *Proc. AAAI-05* (to appear).

Srinivasan, A. (2000). *The Aleph manual* (Tech. Rept.). Computing Laboratory, Oxford University.

Taskar, B., Abbeel, P., & Koller, D. (2002). Discriminative probabilistic models for relational data. *Proc. UAI-02* (pp. 485–492).