Learning in First-Order Probabilistic Representations

Matthew Richardson
Ph.D. General Examination
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
mattr@cs.washington.edu

Abstract

Learning probabilistic models has been an important direction of research in the machine learning community, as has been learning first-order logic models. Ideally, we would like to be able to combine the two, i.e., to learn first-order probabilistic models. Because of their ability to handle uncertainty and compactly model complex domains, these models are the object of growing research interest. This research comprises three main directions: knowledge-based model construction (KBMC), stochastic logic programs (SLPs), and probabilistic relational models (PRMs). This paper surveys these approaches, and suggests opportunities for further research and improvement, particularly with regard to modifying them so they may scale to handle large amounts of training data.

University of Washington, Department of CSE Technical Report #03-10-01

Last updated June 11, 2002.

1. Introduction

Data storage capacity and computer processing power have been growing exponentially for many years. As a result, users wish to use computers to learn and reason over increasingly complicated domains. Such domains frequently have relationships between entities, are probabilistic in nature, and may involve very large amounts of training data. Machine learning in these first-order probabilistic domains is an area of research that is just beginning, but has the potential for great benefits.

Historically, much of machine learning uses flat, propositional, attribute-based models. In these, each instance of data is described by a fixed set of attributes, and the examples are assumed to be independent of each other. In contrast, much of the real-world data collected today is relational (i.e. first-order) in nature – the data consists of entities with their own set of attributes, and also relations between them. For example, to understand the medical history of a family requires explicitly modeling the relationships between people. Without these, important genetic causes of disease may be missed. Another example is a large corporation, which must maintain and track inventory information, involving hundreds or thousands of products, suppliers, and retail stores. Modeling such complex processes requires algorithms that can handle relationships and interactions between various entities in the domain.

The majority of existing data mining and machine learning algorithms are propositional in nature (e.g. decision trees, Bayesian networks, etc.). As a result, one of the most common methods for mining relational data is simply to "flatten"[1] it into a propositional form and then apply standard mining algorithms to it. This approach has many problems. The first is that the size of a relational knowledge base can increase dramatically when flattened. Also, learning a propositional model for what is actually a first-order domain may work on one specific instance of the domain, but the model will not be generalizable. In order to generalize to different instances, the model itself needs to be relational. In the inventory example, a propositional model could learn specific relations between retail stores and each individual supplier, but it would then be very difficult to add or remove suppliers or retail stores from the model.

Most propositional learning algorithms assume the training examples are independent of each other. This is typically a reasonable approximation, but when the data is a result of flattening

[1] By "flatten", we mean to convert relational knowledge into a propositional form. For example, if the knowledge base contains the fact knows(bob,joe), this can be flattened to a single proposition bob_knows_joe. Difficulties arise if the knowledge base contains facts such as knows(bob,X), in which case flattening results in many propositions, one for each person in the knowledge base.

a relational domain, this assumption becomes particularly incorrect. For example, given medical data about a father and son, should the training data consist of two instances (one for the father, which would include the son as an attribute, and then one for the son which would include the father as an attribute) or just one instance (describing the two of them together)? The training set that results from flattening relational data will often contain either duplicated data, or data which is ignored. Either one can lead to statistical difficulties.

Finally, using a first-order model for learning can lead to the discovery of interesting relations, or information about the relations, between entities in the model. It is very difficult to learn such relationships from a propositional model. To do so would require including all possibly interesting properties and all possibly related items as attributes in the training set, which is simply not feasible for reasonably complex domains. For these reasons, it is important to find methods for data mining with first-order reasoning and learning abilities.

The world is not only relational, but also uncertain. Again consider the case of learning about how disease depends on family medical history. Clearly, diseases are very probabilistic in nature. To understand, for example, what family factors may influence a person's probability of contracting cancer, requires the ability to reason and learn probabilistically. Recent developments in probabilistic (propositional) models (such as Bayesian networks) have been very successful, showing the promise that probabilistic modeling has.

Businesses and consumers wish to reason over and understand domains involving complex relationships such as these. Reasoning methods which can handle either relational data *or* uncertainty have been the subject of much study, but it is only recently that algorithms have been designed which learn models that are both first-order *and* probabilistic. This paper surveys the state of the art of these learning methods.

We begin with an overview of logic and probability as it is used in machine learning. Current methods can be roughly divided into the categories shown in Figure 1. The x-axis represents whether or not the model allows uncertainty, while the y-axis whether the model is propositional or relational. We begin with the simplest quadrant: *propositional logic*.

2. Logic and Uncertainty

In propositional logic, terms represent *propositions*: individual statements about the domain (e.g. cat_in_house, bob_is_hungry). Though general purpose reasoning in propositional logic is NP-complete [5], it can be done in polynomial time if the representation is limited to proposi-

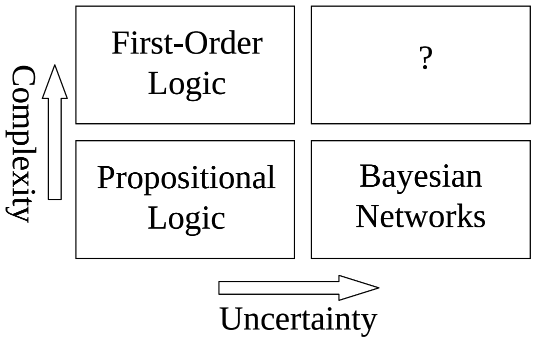


Figure 1: Knowledge representation and reasoning with and without relations or uncertainty

tional *Horn clauses*[2]. Many of the methods introduced in this paper restrict themselves in this way.

Propositional logic has two major limitations. First, propositions are always either fully true or fully false. This limits our ability to model and reason in the real world, where uncertainty is prevalent. Probabilistic models, specifically *Bayesian networks*, overcome this. The second limitation is that modeling a reasonably complex domain can require an exponential number of propositions. For example, to represent who knows whom among 20 friends would require 20·20 propositions (a_knows_b, c_knows_a, …). This limitation can be overcome by using first-order logic, which allows relations to be abstracted as predicates (e.g. knows(X,Y)). As discussed in the introduction, both of these limitations need to be overcome in order to deal with modern, complex data mining domains.

Unfortunately, both options come at a price. Inference in Bayesian networks takes time that is exponential in the size of the largest clique, so for reasonably complex domains we must resort to approximate inference methods instead. Also, reasoning in first-order logic is only semi-decidable. Even so, the benefits of each often outweigh the costs. One may wonder, then, whether it is possible to combine the two into a single model.

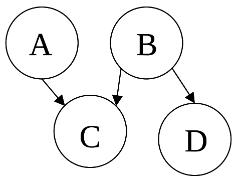
In fact, for years researchers have been studying models that use both probability and first order logic. It is only recently though that methods have been developed to learn them. These models can be divided into roughly three categories: knowledge-based model construction (KBMC), stochastic logic programs (SLPs), and probabilistic relational models (PRMs).

In the next sections, we introduce each of these methods and explain how they are used to perform inference and how their models are learned. We then compare the methods and propose how they may be extended or modified, especially for learning on large data sets. The paper then ends with some related work and concluding thoughts. First, however, we review inference and learning in Bayesian networks and first-order logic, which form the basis for the three methods.

[2] A Horn clause is a disjunction with at most one positive term. More simply, it is an implication with all positive terms, and only one atom in the *head*: $a \wedge b \wedge c \Rightarrow d$ (d is the head, or *consequent*)

Learning with Bayesian Networks

A Bayesian network (BN) [23][41] is a tuple **B**=<**G**,**θ**>, where **G** is a directed acyclic graph of N nodes, and **θ** is a set of parameters. Each node represents a random variable, and edges represent probabilistic influence between nodes. A Bayesian network compactly represents the joint probability distribution $P(X_1,X_2,\ldots,X_N)$. The graph structure defines probabilistic independence relations: a node is probabilistically independent of its non-descendents, given its parents. As a result, the BN needs only to define $P(X_i \mid pa(X_i))$ for each node (where $pa(X_i)$ is the set of parents of node X_i). Typically, this is done with a conditional probability table (CPT)[3]. If the variables are all Boolean, the probabilistic independence reduces the number of parameters needed to represent the full joint probability from $O(2^N)$ to $O(2^{\max_i\{|pa(X_i)|\}})$. Figure 2 shows an example Bayesian network of four variables: A, B, C, and D.



$$P(A,B,C,D) = P(A)P(B)P(C \mid A,B)P(D \mid B)$$

Figure 2 : Example Bayesian Network

Since a BN defines the joint probability distribution, it may be used to ask any probabilistic query of any variables in the network, given the values of any other variables. Doing so is called *inference*, and is NP-hard in general [6]. To speed up inference on large networks, approximate inference techniques such as likelihood weighted sampling or loopy belief propagation [38] are often used.

There are two aspects of a Bayesian network that may be learned: the structure, and the parameters. As will be seen, the distinction between learning structure and learning parameters is one that appears in all probabilistic learning presented in this paper. When learning the parameters, the structure (**G**) is known and constant. The task then is to learn, for each node in the network, a probabilistic model of that variable given the values of its parents. The goal is typically to maximize the likelihood of the training data. If the training data is complete, this is accomplished simply by counting the co-occurrence of the values of the node for the various values of its parents[4]. When some of the data values are missing, the well-known EM algorithm [10] is employed.

[3] For the remainder of this paper, we will always assume that conditional probability distributions are specified using a CPT.

[4] This is true only if we assume each training data instance is independent.

EM estimates the CPTs from the known data, then uses those to estimate the missing values, then uses those to re-estimate the CPTs, and repeats until convergence.

When the structure is unknown, the task is to learn both the parameters and the structure of the Bayesian network. One of the most common methods for this is that of Heckerman et al. [19], which searches through the space of possible networks by greedily adding, removing, or reversing edges in order to maximize a *scoring* function. The scoring function is a Bayesian estimate of the quality of the model, given the data and a prior over possible graph structures.

Note that each time the network changes, the probabilistic parameters must be updated as well. When there is no missing data, this may be done efficiently because edge additions or removals only affect the nodes that are local to the change. When there are missing values, edge changes no longer have a localized effect, theoretically requiring EM to be re-run for each one. Algorithms such as structural EM [11] address this problem by filling in the missing data values with their expected value for each structural iteration. For a more thorough introduction to Bayesian networks, including how they are learned, see [18].

Learning with First-Order Logic

Logic programs and logical inference are useful tools for knowledge representation and reasoning. Here we give only a brief summary of the methods used in first-order logical deduction and induction. For an introduction to first-order logic and inductive logic programming, read [31]. A more thorough approach can be found in [32]. In what follows, we limit our representation to first-order Horn clauses.

In deductive inference, the goal is to determine, given a knowledge base (**KB**) and a statement (**E**), whether the **KB** entails **E** (written $\mathbf{KB} \models \mathbf{E}$). Alternatively, the goal may be to find all statements **E** such that $\mathbf{KB} \models \mathbf{E}$. In either case, deductive methods such as forward-chaining using resolution [46] are used to combine clauses in the knowledge base in order to form new clauses that are entailed by it. Resolution can also be used to prove a statement by *refutation*. If $\mathbf{KB} \wedge \neg \mathbf{E}$ can be resolved to an empty statement, then $\neg\mathbf{E}$ conflicts with **KB**, and thus $\mathbf{KB} \models \mathbf{E}$.

Logic programs may be also learned from examples. Given some background knowledge (**B**), and a set of examples (**E**), the goal is to find a set of statements (**H**, the hypothesis) such that $\mathbf{B} \wedge \mathbf{H} \models \mathbf{E}$. Notice that this is the reverse of logical deduction, and is called *induction*, or *inductive logic programming* (ILP).

Most ILP methods can be classified into one of two categories: *top-down* or *bottom-up*. Top-down approaches, exemplified by FOIL [45], begin with an initially empty hypothesis, which is grown by adding clauses that cover positive examples. The clauses are specialized by adding antecedents such that they cover the maximum number of positive examples while misclassifying a minimum number of negative ones. This process is repeated until the hypothesis completely covers the training set.

In bottom-up approaches, the hypothesis is formed by repeated generalization of the examples. This generalization is done using inductive methods such as inverse resolution [37] (used by Cigol) or mode directed inverse entailment [33] (used by Progol). Inductive methods can be thought of as "running the proof backwards", since their goal is to find the **H** that, if deductive methods were to be used, entails **E**. It is interesting to note that bottom-up ILP has the potential to create concepts not explicitly given to it. Naturally, the disadvantage of being able to create new concepts is the huge search space that results. As a result, bottom-up ILP is usually directed by multiple constraints, such as requiring that all clauses are range-restricted[5].

Bayesian networks are used for inference and learning in propositional, probabilistic domains. In contrast, ILP is used for learning in first-order, non-probabilistic domains. We now introduce the first of the three methods for combining first-order reasoning with probability: *Knowledge-Based Model Construction*.

3. Knowledge-Based Model Construction

One way to combine probability and first-order logic is simply to augment an existing first-order (Horn clause) knowledge base with probabilistic information. This is the approach taken by Knowledge-based model construction (KBMC) methods, as exemplified by Koller and Pfeffer in [27] and by Kersting and De Raedt's *Bayesian Logic Programs* [24][25], which derive from work by Ngo and Haddaway [40] and Wellman et al. [48]. Ng and Subrahmanian's *probabilistic logic programs* [39] are also in a similar vein.

The various KBMC approaches differ in their details, but the basic idea is always the same: with each clause in a knowledge base, we associate a set of parameters that specify how the consequent probabilistically depends on its antecedents. In the simplest case, this is a single parameter that specifies the probability that the consequent holds given that the antecedents hold. In Kersting's Bayesian logic programs (BLPs), a complete conditional probability table (CPT) may be specified.

Table 1 gives an example BLP, and a CPT for one of the clauses. As can be seen, the probability that a person, say 'bob', exercises is 0.8 if he owns a gym membership and is 0.4 otherwise. Because the clause defined for all 'X', this probability distribution is the same for all people

[5] A range-restricted clause is one in which every variable in the head appears at least once in the body.

healthy(X) ← eats_well(X), exercises(X)
eats_well(X) ← eats(X,Y), healthy_food(Y)
exercises(X) ← gym_member(X)

P(exercises)	gym_member
0.8	yes
0.4	no

Table 1: (a) Some Horn clauses defining a simple BLP for a health domain. (b) Example CPT for one of the clauses.

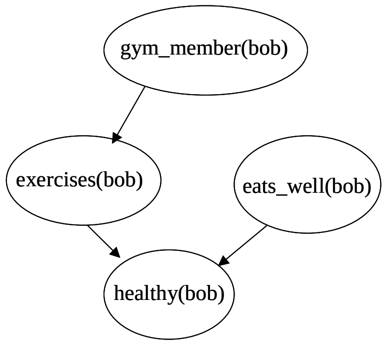


Figure 3: BN result of querying a BLP

in the model. This parameter sharing facilitates both a compact representation, and learning. KBMC handles any Horn clause, allowing relations such as ‘eats(X,Y)’ (meaning person X eats food Y), or even higher-arity relations such as ‘uses(Person, Gym, Machine)’ (which would represent which machines a person uses to exercise).

To answer a query, KBMC converts the knowledge base into a Bayesian network which represents the same probability model. Each grounded predicate that is relevant to the query appears in the Bayesian network as a node. The relevant predicates are found by backward-chaining. For example, Figure 3 shows the Bayesian network that would result from the query “healthy(bob)?” given “eats_well(bob)” and “gym_member(bob)”. The CPTs in the Bayesian network come directly from the probabilistic parameters in the logic program. Once the query has been converted into a Bayesian network, any standard BN inference technique may be used to answer the query.

One complexity arises if there are multiple clauses with the same grounded consequent. For example, consider the second clause in Table 1: eats_well(X) ← eats(X,Y), healthy_food(Y). If

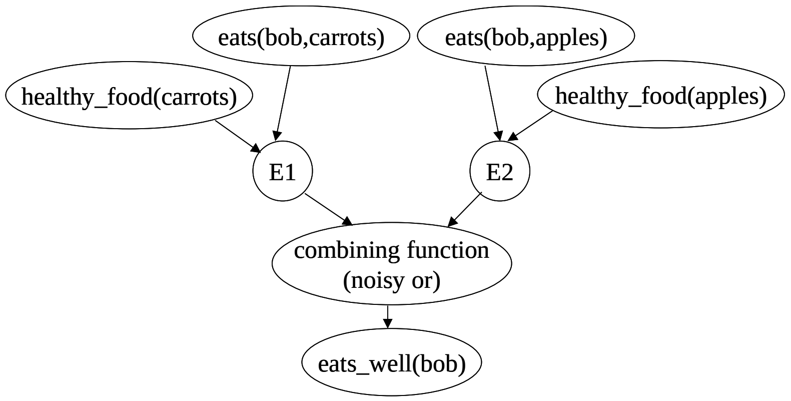


Figure 4: When there are multiple sources of evidence, they are combined using a node which performs a function such as noisy-or. Above is an example Bayesian network for the query “eats_well(bob)?”.

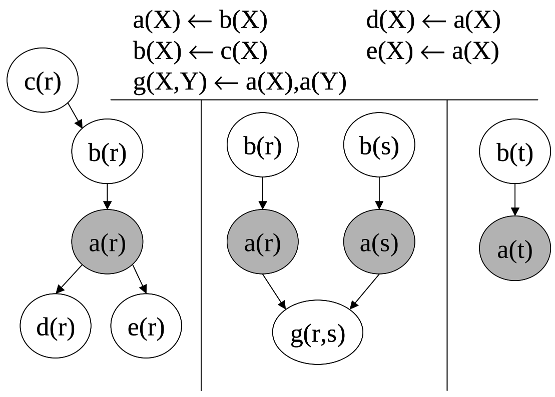


Figure 5: Structure for the clause a(X):-b(X) remains the same regardless of query

eats(bob,carrots) and eats(bob,apples), both of which are healthy_food(.), then what is the probability that eats_well(bob)? For this, KBMC uses a *combining function* (See Figure 4). The combining function merges multiple probabilities into one, and is performed by a set of additional nodes in the Bayesian network. One commonly used combining function is noisy-or, which makes the assumption that the probability that one of the "causes" fails to produce the effect is independent of the success or failure of the other causes. Most commonly occurring combination functions are *decomposable* [4]; any of these may be used.

Learning with KBMC

There are two parts to a KBMC knowledge base: the set of clauses, and the set of probabilistic parameters associated with them. These parts are analogous to the *structure* and *parameters*, respectively, of a Bayesian network. As with BNs, we first consider how to learn the parameters given the set of clauses.

KBMC takes as input a KB (with probabilistic parameters), evidence, and a query, and generates a Bayesian network from them. Though the network may be different for each query, the structure and CPT parameters associated with a clause will always be the same. Figure 5 demonstrates this with a simple logic program and the BN that results from three different queries. Each node in the network is associated with at most one clause, and can be viewed as a separate "experiment" for it. As with Bayesian networks, the counts for each clause are simply accumulated across the training examples in order to compute the maximum likelihood parameterization for it.

More details, along with a proof that this is correct both with complete training data and without (in which case EM is employed), can be found in [27][6].

Though research on learning the "structure" (set of clauses) for KBMC has not yet been done, Kersting et al. [25] propose using methods from inductive logic programming. We believe that a heuristic search technique, such as that used for learning the structure of Bayesian networks, would also work.

We now discuss stochastic logic programs, the second method for combining probability with first order reasoning.

4. Stochastic Logic Programs

As with KBMC, a stochastic logic program (SLP) [34] [36] consists of a set of first-order clauses, augmented with probabilities. However, the semantics of an SLP are very different from those of KBMC. In KBMC, inference determines a probability distribution for the value of a grounded predicate (e.g. What is the probability of healthy(bob)?). With an SLP, "inference" results in a probability distribution over the possible groundings of a given predicate (e.g. for healthy(X), what is the probability that X=bob vs. X=joe).

A stochastic logic program consists of a set of *stochastic clauses*, each of which has the form p:C, where C is a first-order range-restricted Horn clause, and p is a probability. Note, in this paper we consider only normalized, pure SLPs. That is, SLPs that have a probability associated with all clauses, and in which the probabilities for any given predicate sum to 1. For a more thorough treatment, including methods for learning unnormalized and/or impure SLPs, we refer the reader to [8].

SLPs are like a first-order extension of stochastic context-free grammars (CFGs) [30]. A stochastic CFG is a CFG in which each production rule has an associated probabilistic parameter p. The parameters provide a probability distribution over production rule firings, and hence affect the distribution of terminal symbols. Similarly, in an SLP, the p parameters determine the probability distribution over first-order clause rewritings, and hence affect the distribution of grounded predicates.

Inference is performed by SLD-refutation [32]. Figure 6 shows an example SLP and the associated SLD-tree for the goal "s(X)?". The SLD-tree consists of all refutations of a goal via resolution. Each branch is a result of resolving with different clauses, and each path through the tree

[6] Note that, although this learning process involves constructing a network for each training example, these networks only need to be constructed once. When there are missing values in the training set, the time spent constructing the BNs is insignificant compared to the time spent iterating with EM [27].

0.4 : s(X) ← p(X), p(X). **0.6** : s(X) ← q(X).	**0.3** : p(a). **0.7** : p(b).	**0.2** : q(a). **0.8** : q(b).

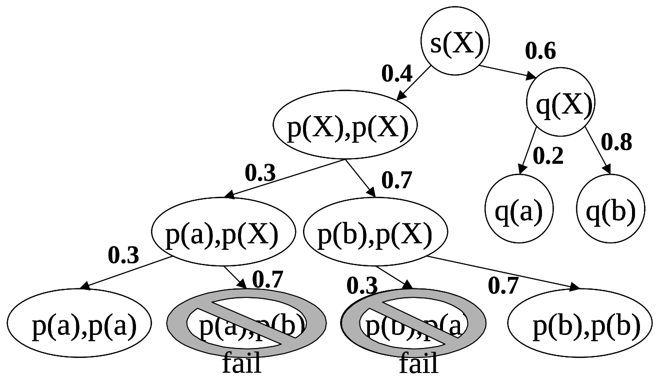


Figure 6: (a) Example SLP and (b) Associated SLD-tree for s(X)

represents one complete refutation (or failure to refute). The probability of a path is the product of the probabilities of the clauses used at each branch. These probabilities are accumulated for each grounding of the query predicate:

P(s(**a**)) ∝ 0.4·0.3·0.3 + 0.6·0.2 = 0.156	P(s(**b**)) ∝ 0.4·0.7·0.7 + 0.6·0.8 = 0.676

To find the true probabilities, these are normalized, which results in P(s(**a**))=0.1875, and P(s(**b**))=0.8125. The normalization is necessary if any of the paths in the SLD-tree end in failure. Note that, in general, the SLD tree may be exponential in the number of nodes in the network.

SLPs can be used to represent a variety of probabilistic models, such as Bayesian networks and Markov random fields. To build these, each variable is represented by a predicate with at least one parameter, which takes value 1 or 0, indicating that the variable is true or false respectively. By computing a probability distribution over possible groundings of the predicate, SLP inference indirectly computes the probability that the variable is true and the probability that it is false. For example, to encode the predicate healthy(*X*), we would use the predicates healthy(0,*X*) and healthy(1,*X*) instead. The probability that healthy(*bob*) is true would then be:

$$\frac{P(\text{healthy}(1, bob))}{P(\text{healthy}(1, bob)) + P(\text{healthy}(0, bob))}$$

Learning with SLPs

As with BNs and KBMC, we distinguish between learning the structure, and learning the parameters, of a SLP. To learn the structure, Muggleton [35] uses standard ILP techniques, such as his Progol4.5 [33], to induce a logic program as a basis for the SLP. As with Bayesian networks, this depends on a scoring function that includes a discount term based on the complexity of the model. Though the scoring function is similar, the search only greedily adds clauses to the program, and hence may result in a sub-optimal result. The clauses in the induced logic program are then augmented with uniform probabilities to create an SLP.

To learn the parameters of the SLP, Muggleton proposes a simple method which simply re-estimates the probability labels by counting the fraction of times each clause is used in a successful derivation of the goal. This simple strategy does increase the posterior probability of the model, but is sub-optimal in that it does not attempt to search for nor does it find the parameter settings which result in the highest posterior probability. It will deduce the optimal parameter settings only in the case where each positive example has a unique derivation in the logic program (a scenario we believe is fairly uncommon).

Cussens [8] summarizes other techniques that have been developed for parameter estimation of SLPs. One such technique is *Improved Iterative Scaling* [9], an algorithm that iteratively updates the parameter estimates until they converge. Riezler [47] has extended this to the case where data is incomplete, using a method analogous to EM. When the SLP is normalized, Cussens' *Failure-adjusted maximization* algorithm may be applied.

So far, we have presented two techniques for combining probability and first-order logic. Though the semantics are different, both techniques are based on a set of clauses augmented by some probabilistic information. We now present the third major technique, *Probabilistic Relational Models* (PRMs), which take a different approach. Instead of adding probability to first-order logic models, PRMs add first-order logic to probabilistic models. Specifically, they add objects and relationships to Bayesian networks.

5. Probabilistic Relational Models

Bayesian networks have been a very successful tool for reasoning in probabilistic domains. However, they do have limitations. As a result of their propositional nature, the entirety of a domain must be known in advance, and probabilistic parameters which could be shared may end up scattered across many CPTs. For example, consider using a Bayesian network to model the processing involved in a computer chip manufacturing plant. Such plants have many machines, each of which may be affected by a variety of variables, such as the temperature of the room, the cleanliness of the air, etc. Though some of the machines may be identical, in a (propositional) Bayesian network each must be represented by its own set of nodes, with their own CPTs describing how the machine is influenced. This redundancy can have a negative effect on the efficiency of inference, as well as on the ability to learn such models due to the increased number of pa-

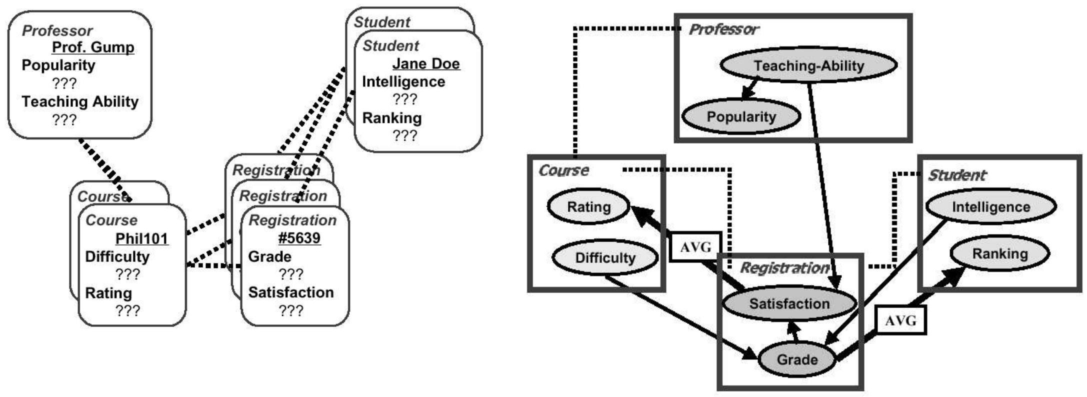


Figure 7: Example PRM for a University domain

rameters. Furthermore, using this network to model the processing at a different plant would require adding and/or removing nodes due to machine differences between the two.

What is needed is an abstract concept of a machine, which can be instantiated multiple times (or none at all), for each type of machine in the plant. Each object would have various properties, and also relations to other objects it affects (or is affected by). Such a model was proposed by Friedman et al in [12] and is called a *Probabilistic Relational Model* (PRM). The PRM is a refinement of the probabilistic frame-based systems introduced by Koller and Pfeffer in [28].

A PRM consists of a set of classes $\{X_1,X_2,\ldots,X_n\}$. Each class X has a set of attributes $\mathbf{A}(X)$; each attribute A is denoted $X.A$ (For example, Person.*height*). Each class also has a set of reference slots $\mathbf{R}(X)$, where each reference slot ρ is denoted $X.\rho$, and points to an instance of the same or another class (For example, Person.*mother*). Reference Slots can be composed to form a *slot-chain* (For example, Person.*mother*.*father* refers to a person's maternal grandfather).

A PRM also defines the probabilistic relationship between attributes of classes. An attribute may depend on any attribute of the same class, or of a class that is reachable through some slot-chain. Figure 7 shows an example PRM [14]. Reference slots are denoted by dotted lines between classes (e.g. Registration.*course* and Registration.*student*). As can be seen in the figure, PRMs allow relations as simple as "A professor's popularity depends on his teaching ability" or as complex as "A student's grade in a class depends on the difficulty of the class and the intelligence of the student". Note that slot-chains are not always 1-to-1. In the example PRM, a student's ranking depends on the grade he receives in a course, but since there are multiple courses, there must be some way to combine them when determining the ranking of the student. For this, PRMs use the concept of an *aggregate function*, which is very similar to the concept of a combining function in KBMC. An aggregate function combines a set of attributes into a single value. In this example, the student's ranking depends probabilistically on the average of the grades he received.

A PRM can be thought of as a template which, when given a specific domain of objects, is "compiled" into a Bayesian network. Given a PRM and a set of objects, inference is performed similarly to KBMC, using backward-chaining to determine the relevant objects and attributes, and constructing a Bayesian network on which standard inference techniques are used.

Learning with PRMs

As before, learning in PRMs can be divided into two tasks: learning the structure of the model, and learning the parameters. For PRMs, the "structure" is the relational skeleton, which specifies the parent set pa($X.A$) for all attributes A and all classes X.

When the structure is known (along with the set of all objects in the domains and the relationships between them), learning the parameters is nearly identical to learning the parameters of a standard Bayesian network. The only difference is that some parameters are *tied*, or forced to take the same value. The resulting likelihood function still decomposes into a product of terms which may be individually maximized, as in BNs, using simple counts.

Structure learning is also similar to BNs. A hill-climbing procedure searches through the space of possible structures using operations such as adding, deleting, or reversing edges in the PRM. The search is directed by a model scoring function, as with standard Bayesian networks. For more details, such as ensuring that the PRM always induces an acyclic BN, see [14].

Structural Uncertainty: PRM 2

A standard PRM has full knowledge of the relations between objects. In [15] and [16], Getoor relaxes this requirement by introducing the *Probabilistic Relational Model II* (PRM2). A PRM2 allows two types of uncertainty over the relationships between objects: *reference uncertainty*, and *existence uncertainty*.

In *reference uncertainty*, the objects in the domain are known, but the references between them are not (though the number of relations is known). For each unknown reference slot, the PRM2 induces a distribution over objects, which determines the probability that the given object is assigned to that slot. However, because the PRM2 must be able to generalize for any set of objects, it cannot directly specify a probability distribution over the objects of the training set. Instead, it indirectly selects objects according to their attributes.

For each reference slot, objects are first partitioned according to some of their attributes. Then, from the training set, the PRM2 learns a probability distribution over partitions. This distribution is just like any other attribute, in that it is conditioned on some other set of attributes, which may be in the same class or in some other class via a slot chain (these can be thought of as the parents of the slot). Each slot has a CPT which specifies the probability distribution over partitions, given its parents. The probability mass assigned to a partition is then uniformly distributed over the objects contained within.

For example, consider the university domain given above in Figure 7. The relation Course.*taught-by* may be uncertain. The PRM2 defines some set of attributes, say {*department,seniority*}, by which to partition all of the professors (not shown in the figure). The probability distribution over departments is specified by a CPT which may have as parents, for instance, Course.*subject* and Course.*difficulty*. From the training set, the PRM2 may learn, for example, that the Course.*taught-by* relation for easy computer courses is more likely to be filled by new professors in computer science than by senior professors in linguistics.

If the partition attributes for each relation are provided, then a PRM2 may be learned in the same manner as a standard PRM (with only slight modifications). However, the partition attributes are not assumed to be known in advance. To find them, the hill-climbing structural search procedure is augmented with two new actions: *refine* and *abstract* which add or remove (respectively) an attribute from the partition set for a relation (which is initially empty).

The second type of structural uncertainty introduced in PRM2s is *existence uncertainty*. Recall, in reference uncertainty, the number of relations is known, but the relations between objects are not. In existence uncertainty, not even the number of relations is known. To handle this, Getoor et al. augment classes with an existence (E) attribute, which is *true* if the object exists and *false* if it does not. *X.E* is treated as any other attribute, in that it may have parents, and is associated with some CPT. The result is a probabilistic model for the existence of relations between objects.

For example, consider the student domain and suppose that the registration objects are undefined. The Registration.*E* attribute may have, as parents, the student's major and year of study, and also the course subject. The associated CPT would likely represent that a registration object is more likely to exist if the major and year of study of the student match the department and difficulty of the course. The model (conceptually) instantiates all possible registration objects, assigning to each a probability of existence based on this.

X.E is treated like any other attribute of the class, so learning a PRM2 with existence uncertainty is also a simple modification of the standard PRM algorithm. Care must be taken when estimating the probabilities associated with non-existent objects, but otherwise the algorithm remains the same.

PRMs provide a useful framework for modeling probabilistic, relational domains, such as those that are often found in the real world. Though PRMs can represent only a subset of first-order logic, their basis in objects and relationships are a natural fit to the way in which domains are often understood and described. As a result, they are a useful tool for modeling complex, real-world domains.

In the following sections, we discuss learning in first-order probabilistic representations in general, and compare the three techniques presented above: KBMC, SLPs, and PRMs.

6. Overview and Comparison

Independence: One of the problems with learning in first-order probabilistic representations is that in order to calculate the likelihood of the model, the training examples are typically assumed to be independent of each other. Nearly all probabilistic learning methods make this assumption, but in first-order domains it is more likely to be false since the data instances are all (potentially) related to each other [27]. Though the learning methods presented earlier currently make this independence assumption, there may be methods to overcome this. For instance, for KBMC and PRMs, the correct method for learning would be to build one, incredibly large Bayesian network which incorporates all of the training examples. Though impractical as stated, it may be possible to optimize or approximate this process. For instance, with a simple assumption about limited propagation of effects, it may suffice to model small clusters of training examples, and assume that each cluster is independent. A more principled method might calculate the effect that training examples have on each other and introduce this effect via a set of periphery nodes when constructing the network for each example. It may be possible to simulate the full network exactly by continually reconstructing (or destroying) portions of the network as they are needed (or not) by the inference and learning procedures. It is plausible that one of these methods will outperform a learning algorithm that assumes that all examples are independent. More research in this direction should be done to at least investigate the effect that this independence assumption has.

Combining Functions: With first-order models, the number of objects influencing a given object is not known in advance. In non-probabilistic reasoning, this is not a problem – more sources of evidence do not make a fact “more true”. In probabilistic reasoning, there must be some way to combine the influences. Because of this, KBMC, PRMs, and SLPs all have a mechanism for combining influence. In the case of KBMC and PRMs, this is done explicitly through a combining function (KBMC) or aggregation function (PRMs). In SLPs, combination is done implicitly as a weighted average, which results from having multiple clauses with different weights for the same head term. We believe that separating the combining function from the program (as is the case with KBMC and PRMs) assists in learning. A learning algorithm that explicitly knows about (and possibly chooses from) combination functions may be able to better fit the training data than one for which the combining function is learned implicitly as part of determining the clause weights. Some combining functions can not be represented as a weighted average, in which case the SLP learning algorithm would have to create intermediate "hidden" predicates, further complicating the learning procedure.

Since the function used to combine probabilities is so important, it would be useful to further research them. It would be interesting to investigate methods for automatically learning the appropriate combination functions, rather than forcing them all to be the same, pre-defined function. It would also be interesting to research more complicated methods for handling multiple sources of evidence. For instance, there may be some way to cluster the sources of evidence, so that they are not all considered equally predictive, or to allow the different clusters to affect the consequent in different ways. Some of the work from Jaeger on relational Bayesian networks [21] may be relevant. In this work, Jaeger allows the specification of nested combination functions which allow multiple sources of evidence to be divided into partitions, using one function to internally combine sources of evidence (for instance, linearly), while the partitions themselves are combined in another (for instance, as a noisy-or).

Probabilistic at heart: In order to represent probabilistic information with SLPs, predicates must be augmented with an additional Boolean parameter. Though SLPs can be used to ask interesting queries, as Cussens [7] points out, using them naïvely for probabilistic modeling is very inefficient. It is not clear that an efficient algorithm can be found. We believe that because KBMC and PRMs are built explicitly for modeling probabilistic phenomena, it will be easier to make approximations and find optimizations for them than for SLPs. As a result of being probabilistic "at heart", KBMC and PRMs seem to be a more appropriate choice for probabilistic first-order reasoning and learning.

Structure Learning: It is interesting to note that there are two main ways that structure is learned. The first is inductive logic programming (ILP) methods such as inverse resolution. The second is hill-climbing search (adding, removing, and reversing edges) to maximize a heuristic scoring function. Each method has advantages. It would be interesting to compare the performance of each when applied to KBMC and PRMs. Kersting proposed learning KBMC structure by ILP, but it would also be possible to apply PRM (and BN) structure learning techniques instead. PRMs learn structure by hill-climbing to maximize a scoring function, but it would also be possible to represent the information as a logic program and learn the structure with ILP instead. In either case, lessons learned from one method may be applicable to the other. For instance, it may be easier to incorporate prior knowledge with ILP methods, since prior knowledge may simply be stated with additional first-order clauses. However, hill-climbing search is easy to modify to incorporate a wider variety of search tasks, as was demonstrated with PRM2s (recall, in a PRM2, the partition attributes for reference uncertainty are learned by simply augmenting the hill-climbing search procedure with the ability to add or remove attributes from the partition set). Research on the differences and similarities between these methods for learning structure could potentially result in hybrid techniques, such as one that uses a hill-climbing search by adding and removing edges, but does so in a more directed fashion, using individual training instances to determine search direction (a la inverse resolution).

7. Scaling to Large Data Sets

Modern data mining and machine learning domains are not only more complex than before, but also are much larger. Even though computer processing speed has been growing exponentially, the size of the data sets which researchers wish to mine have been growing even quicker. As a result, it is worth investigating how these first-order probabilistic learning algorithms will scale when used on large data sets.

Missing data: The first item to note is that the scalability of the algorithms depends strongly on whether or not there are missing values in the training set. When the training data is complete, parameter estimation reduces to simple counting for PRMs and KBMC. In most situations, SLPs would still require an iterative update algorithm unless the user is willing to settle for non-optimal parameter settings. For structure learning, if the training set is complete then PRM structure learning (as with BN structure learning) decomposes into a fairly efficient search since each change in the network has only a small local effect.

This is not the case when there are missing values. When the training data is not complete, EM is typically employed. Using EM drastically increases the runtime required for learning on a large data set. For both PRMs and KBMC, EM may be used without modification when learning parameters. SLPs use a similar technique, called *Iterative Maximization* [47]. For learning structure, an algorithm such as structural EM [11] may be useful for speeding up learning. It would certainly be interesting to apply structural EM to the problem of learning the structure for KBMC and SLPs. As a final note, if the absent values are missing at random, and the data set is large enough, then the missing values may be safely ignored. For very large data sets, this would result in dramatic speed improvements over EM.

Data access: One major difficulty with large data sets is that a learning algorithm must carefully limit its data accesses, otherwise it may incur so much disk overhead as to make learning infeasible. One technique to overcome this is to perform the hill-climbing search "in parallel", so that all potential model changes are examined with only one pass over the training set [1]. This gives rise to other difficulties, such as limited RAM. Hulten and Domingos [20] present a general framework for scaling up learning algorithms in this way, and show specifically how their methods can be applied to learning the structure of Bayesian networks. Their methods may be directly applied to structure learning in PRMs. With some modification, they also may be applicable to learning KBMC structure. A method similar to AD-trees [29], modified to handle first-order data, may also prove useful for speeding up learning for large data sets. AD-trees pre-calculate and store most of the sufficient statistics needed for learning algorithms such as the hill-climbing search used by PRMs.

Incremental Learning: Sometimes, training data arrives as a stream. One way for dealing with such a data set, or with very large data sets in general, is to employ incremental learning methods. It would be interesting to explore both incremental ILP and incremental Bayesian network learning methods, as both pose potential improvements for SLPs, KBMC, and PRMs.

Sparse Candidate: Since KBMC and PRMs can be represented and learned using Bayesian networks, we can also apply various techniques speeding up BN learning. One of these, the sparse-candidate algorithm [13], was already proposed to be used in PRMs [12], and could also be used in KBMC. For a Bayesian network of N nodes, a standard hill-climbing search would consider $O(N^2)$ edge changes (one for each pair of nodes). In the sparse-candidate algorithm, this is reduced to O(kN) by limiting the possible parents of a node to be one of a pre-determined set, of size k. This set is chosen using a measure such as mutual-information between nodes. As long as the parent sets are correct, the sparse-candidate algorithm will find the same network structure as standard hill-climbing would, but in much less time (of course, the parent sets do not always contain the correct parents).

Specialized first-order methods: For both KBMC and PRMs, inference is performed on a propositional Bayesian network structure. We believe that algorithms which instead work directly on the first-order structure of a model should be able to perform inference and learning more quickly. It seems likely that an algorithm that is explicitly designed to work on these specialized, first-order models will be easier to optimize, be more compact, and be more efficient than generic propositional algorithms. For example, compiling the PRM into a Bayesian network may result in a huge BN. There may be large savings in memory and computational requirements if the inference is instead performed directly using the PRM structure, only dividing and propositionalizing nodes whose probability distributions diverge.

PRMs are more constrained and structured than KBMC-style logic programs. Structurally, KBMC allows arbitrary predicates of any arity while PRMs essentially only allow unary and binary predicates (note, however, that higher arity predicates may be simulated via multiple binary predicates). Also, unlike general first-order logic, in a PRM, relations may be composed only along slot-chains. Since they operate only over a restricted subset of first-order logic, and contain additional structure, it seems likely that inference PRMs may be more efficient than inference with KBMC. Pfeffer's research into systems such as SPOOK [42] has shown that algorithms which take advantage of the properties of an object-oriented system, such as PRMs, can lead to incredible speedups in inference. These speedups come from exploiting the structural features in the model such as object encapsulation and well-defined interfaces between objects. These features allow the algorithm to re-use computation for increased efficiency. Though it has been shown that there will likely be no inference techniques with a better asymptotic worst-case behavior for KBMC and PRMs [22], we believe that it may be possible to improve the average case, or at least reduce runtimes by a constant factor. Further, there may be approximate methods applicable to the first-order representation that tradeoff a small reduction in accuracy for a large improvement in runtime.

General Techniques: Finally, there are many general purpose techniques for scaling algorithms to large data sets which may be applicable to first-order probabilistic models. For instance, in all three representations, we could use sampling to reduce the size of the training set. Another technique for scaling up learning involves making simple approximations which allow for faster computation, or a more decomposable search. Also, in many cases, the amount of data needed is significantly less than the amount of data in the training set. Hulten and Domingos derive a formula to determine when a learning algorithm may stop early and declare that the model it has learned is very likely equivalent to the model it would learn on infinite data [20]. The result of this is that a search may be drastically sped up by using only the minimum amount of data needed to provide this guarantee.

8. Other Related Work

Models which combine probability and first-order logic have been the subject of research for many years. Early work by Halpern [17] formalizes the semantics of *statistics* vs. *degree of belief* for probability in first-order logics. Bacchus et al. [3] further expand on degree of belief semantics, and show that they lead naturally to desired properties with default reasoning. Poole further investigates the relation between logical and probabilistic reasoning in [43] and [44].

The work on probabilistic relational models and frame-based systems is rooted in Koller et al.'s earlier work on a probabilistic description logic, P-CLASSIC [26]. In a typical description logic, the only allowed queries are subsumption or membership of objects and sets of objects. P-

CLASSIC allows probabilistic subsumption queries, which are answered with a probability (representing the degree of overlap between classes), rather than a simple "yes" or "no" as would typically be the case.

Also related are Anderson et al.'s *relational Markov models* (RMMs), in which the states of the Markov model are labeled with parameterized predicates [2]. By using a first-order representation for the state, RMMs are, among other things, better able to use smoothing to combat data scarcity.

9. Conclusions

As computational power grows, the complexity of the domains that can be learned also increases. In order to model most real-world phenomenon, methods must be developed which are able to handle both uncertainty, and relationships between objects in the world. Knowledge-based model construction, stochastic logic programs, and probabilistic relational models all seek to bring together probabilistic and first-order reasoning in order to model complex, real-world situations. Though each has its own advantages and disadvantages, improvements will be needed in order to use these methods for learning on large data sets.

10. References

[1] R. Agrawal, T. Imielinski, and A. Swami (1993). Database mining: A performance perspective. In *Special Issue on Learning and Discovery in Knowledge-Based Databases*, 5(6):914-25. IEEE.

[2] C. Anderson, P. Domingos, and D. Weld (2002). 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*, to appear.

[3] F. Bacchus, A. Grove, J. Halpern, and D. Koller (1997). From statistical knowledge bases to degrees of belief. In *Artificial Intelligence,* 87:75-143**.**

[4] C.E. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller (1996). Context-specific independence in Bayesian networks. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*

[5] S. A. Cook (1971). The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pp.151-8.

[6] G.F. Cooper (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393-405.

[7] J. Cussens (1999). Loglinear models for first-order probabilistic reasoning. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence*, pp.126-33.

[8] J. Cussens (2001). Parameter estimation in stochastic logic programs. In *Machine Learning* 43(3):245-71.

[9] S. Della Pietra, V. Della Pietra, and J. Lafferty (1997). Inducing features of random fields. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*19(4):380-93.

[10] A. P. Dempster, N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1-38.

[11] N. Friedman (1998). The Bayesian structural EM algorithm. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*.

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

[13] N. Friedman, I. Nachman, and D. Peér (1999). Learning Bayesian network structure from massive datasets: The "sparse candidate" algorithm. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pp. 206-15.

[14] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer (2001). Learning probabilistic relational models. In *Relational Data Mining*, S. Dzeroski and N. Lavrac, Eds., Springer-Verlag (to appear).

[15] L. Getoor, N. Friedman, D. Koller, and B. Taskar (2001). Learning probabilistic models of relational structure. In *Proceedings of the Eighteenth International Conference on Machine Learning*.

[16] L. Getoor, D. Koller, B. Taskar, N. Friedman (2000). Learning probabilistic relational models with structural uncertainty. In *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*.

[17] J. Halpern (1990). An analysis of first-order logics of probability. In *Artificial Intelligence*, 46:311-350.

[18] D. Heckerman. A tutorial on learning with Bayesian networks (1996). *Technical Report MSR-TR-95-06*, Microsoft Research, Redmond, Washington.

[19] D. Heckerman, D. Geiger, and D. M. Chickering (1995). Learning Bayesian networks: The combination of knowledge and statistical data. In *Machine Learning*, 20:197-243.

[20] G. Hulten and P. Domingos (2002). Mining complex models from arbitrarily large databases in constant time. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, to appear.

[21] M. Jaeger (1997). Relational Bayesian Networks. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence*.

[22] M. Jaeger (2000). On the complexity of inference about probabilistic relational models. In *Artificial Intelligence*, 117:297-308.

[23] F. Jensen (1996). *An introduction to Bayesian networks*. Springer, New York.

[24] K. Kersting and L. De Raedt (2000). Bayesian logic programs. In *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pp.138-55.

[25] K. Kersting, L. De Raedt, and S. Kramer (2000). Interpreting Bayesian logic programs. In *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*.

[26] D. Koller, A. Levy ,and A. Pfeffer (1997). P-CLASSIC: A tractable probabilistic description logic. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence*.

[27] D. Koller and A. Pfeffer (1997). Learning probabilities for noisy first-order rules. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*.

[28] D. Koller and A. Pfeffer (1998). Probabilistic frame-based systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 580-7.

[29] P. Komarek and A. Moore (2000). A dynamic adaptation of AD-trees for efficient machine learning on large data sets. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pp.495-502.

[30] K. Lari and S. J. Young (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35-56.

[31] N. Lavrac and S. Dzeroski (1994). *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, Chichester.

[32] J. Lloyd (1987). *Foundations of Logic Programming*. Springer-Verlag, Berlin. Second edition.

[33] S. Muggleton (1995). Inverse entailment and Progol. *New Generation Computing* 13:245-286.

[34] S. Muggleton (1996). Stochastic logic programs. In *Advances in Inductive Logic Programming*, pp.254-64.

[35] S. Muggleton (2000). Learning stochastic logic programs. In *Proceedings of the AAAI2000 Workshop on Learning Statistical Models from Relational Data*.

[36] S. Muggleton (2000). Semantics and derivation for stochastic logic programs. In *Proceedings of the UAI2000 workshop on Knowledge-Data Fusion*.

[37] S. Muggleton and W. Buntine (1988). Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th International Machine Learning Workshop*. pp.339-52.

[38] K.P. Murphy, Y. Weiss, and M. Jordan (1999). Loopy belief propagation for approximate inference: an empirical study. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence*.

[39] R. Ng and V.S. Subrahmanian (1992). Probabilistic logic programming. In *Information and Computation* 101(2):150-201.

[40] L. Ngo and P. Haddawy (1997). Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147-77.

[41] J. Pearl (1988). *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, San Francisco.

[42] A. Pfeffer, D. Koller, B. Milch, and K. Takusagawa (1999). SPOOK: A system for probabilistic object-oriented knowledge representation. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence*, pp. 541-50.

[43] D. Poole (1992). Logic programming, abduction, and probability. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'92),* pp.530-538.

[44] D. Poole (1993). Probabilistic Horn abduction and Bayesian networks. In *Artificial Intelligence*, 64(1):81-129.

[45] J. Quinlan (1990). Learning logical definitions from relations. *Machine Learning*, 5(3):239-266.

[46] J. Robinson (1965). A machine-oriented logic based on the resolution principle. In *Journal of the Association for Computing Machinery*, 12:23-41.

[47] S. Riezler (1998). *Probabilistic constraint logic programming*. Ph.D. thesis, Universitat Tubingen. AIMS Report 5(1).

[48] M. Wellman, J. Breese, and R. Goldman (1992). From knowledge bases to decision models. In *The Knowledge Engineering Review*, 7(1):35-53.