Learning Stochastic Logic Programs

Stephen Muggleton
Department of Computer Science,
University of York,
York, YO1 5DD,
United Kingdom.

Abstract

Stochastic Logic Programs (SLPs) have been shown to be a generalisation of Hidden Markov Models (HMMs), stochastic context-free grammars, and directed Bayes' nets. A stochastic logic program consists of a set of labelled clauses p:C where p is in the interval [0,1] and C is a first-order range-restricted definite clause. This paper summarises the syntax, distributional semantics and proof techniques for SLPs and then discusses how a standard Inductive Logic Programming (ILP) system, Progol, has been modified to support learning of SLPs. The resulting system 1) finds an SLP with uniform probability labels on each definition and near-maximal Bayes posterior probability and then 2) alters the probability labels to further increase the posterior probability. Stage 1) is implemented within CProgol4.5, which differs from previous versions of Progol by allowing user-defined evaluation functions written in Prolog. It is shown that maximising the Bayesian posterior function involves finding SLPs with short derivations of the examples. Search pruning with the Bayesian evaluation function is carried out in the same way as in previous versions of CProgol. The system is demonstrated with worked examples involving the learning of probability distributions over sequences as well as the learning of simple forms of uncertain knowledge.

Introduction

Representations of uncertain knowledge can be divided into a) procedural descriptions of sampling distributions (eg. stochastic grammars (Lari & Young 1990) and Hidden Markov Models (HMMs)) and b) declarative representations of uncertain statements (eg. probabilistic logics (Fagin & Halpern 1989) and Relational Bayes' nets (Jaeger 1997)). Stochastic Logic Programs (SLPs) (Muggleton 1996) were introduced originally as a way of lifting stochastic grammars (type a representations) to the level of first-order Logic Programs (LPs). Later Cussens (Cussens 1999) showed that SLPs can be used to represent undirected Bayes' nets (type b representations). SLPs are presently used (Muggleton 2000) to define distributions for sampling within Inductive Logic Programming (ILP) (Muggleton 1999a).

Copyright © 2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Previous papers describing SLPs have concentrated on their procedural (sampling) interpretation. This paper first summarises the semantics and proof techniques for SLPs. The paper then describes a method for learning SLPs from examples and background knowledge.

The paper is organised as follows. Section introduces standard definitions for LPs. The syntax, semantics and proof techniques for SLPs are given in Section . Incomplete SLPs are shown to have multiple consistent distributional models. Section introduces a framework for learning SLPs and discusses issues involved with construction of the underlying LP as well as estimation of the probability labels. An overview of the ILP system Progol (Muggleton 1995) is given in Section . Section describes the mechanism which allows user-defined evaluation functions in Progol4.5 and derives the user-defined function for learning SLPs. Worked examples of learning SLPs are then given in Section . Section concludes and discusses further work.

LPs

The following summarises the standard syntax, semantics and proof techniques for LPs (see (Lloyd 1987)).

Syntax of LPs

A variable is denoted by an upper case letter followed by lower case letters and digits. Predicate and function symbols are denoted by a lower case letter followed by lower case letters and digits. A variable is a term, and a function symbol immediately followed by a bracketed n-tuple of terms is a term. In the case that n is zero the function symbol is a constant and is written without brackets. Thus $f(g(X), h)$ is a term when f, g and h are function symbols, X is a variable and h is a constant. A predicate symbol immediately followed by a bracketted n-tuple of terms is called an atomic formula, or atom. The negation symbol is: $\neg$. Both a and $\neg a$ are literals whenever a is an atom. In this case a is called a positive literal and $\neg a$ is called a negative literal. A clause is a finite set of literals, and is treated as a universally quantified disjunction of those literals. A clause is said to be unit if it contains exactly one atom. A finite set of clauses is called a clausal theory and is treated as a conjunction of those clauses. Literals, clauses, clausal

المنارة للاستشارات
www.manaraa.com

theories, True and False are all well-formed-formulas (wffs). A wff or a term is said to be ground whenever it contains no variables. A Horn clause is a clause containing at most one positive literal. A definite clause is a clause containing exactly one positive literal and is written as $h \leftarrow b_1, .., b_n$ where h is the positive literal, or *head* and the b_i are negative literals, which together constitute the *body* of the clause. A definite clause for which all the variables in the head appear at least once in the body is called range-restricted. A non-definite Horn clause is called a goal and is written $\leftarrow b_1, .., b_n$. A Horn theory is a clausal theory containing only Horn clauses. A definite program is a clausal theory containing only definite clauses. A range-restricted definite program is a definite program in which all clauses are range-restricted.

Semantics of LPs

Let $\theta = \{v_1/t_1, .., v_n/t_n\}$. θ is said to be a substitution when each v_i is a variable and each t_i is a term, and for no distinct i and j is v_i the same as v_j. Greek lowercase letters are used to denote substitutions. θ is said to be ground when all t_i are ground. Let E be a wff or a term and $\theta = \{v_1/t_1, .., v_n/t_n\}$ be a substitution. The instantiation of E by θ, written $E\theta$, is formed by replacing every occurrence of v_i in E by t_i. $E\theta$ is an instance of E. Clause C θ-subsumes clause D, or $C \preceq D$ iff there exists a substitution *theta* such that $C\theta \subseteq D$.

A first-order language L is a set of wffs which can be formed from a fixed and finite set of predicate symbols, function symbols and variables. A set of ground literals I is called an L-interpretation (or simply interpretation) in the case that it contains either a or $\neg a$ for each ground atom a in L. Let M be an interpretation and $C = h \leftarrow B$ be a definite clause in L. M is said to be an L-model (or simply model) of C iff for every ground instance $h' \leftarrow B'$ of C in L $B' \subseteq M$ implies $h' \in M$. M is a model of Horn theory P whenever M is a model of each clause in P. P is said to be satisfiable if it has at least one model and unsatisfiable otherwise. Suppose L is chosen to be the smallest first-order language involving at least one constant and the predicate and function symbols of Horn theory P. In this case an interpretation is called a Herbrand interpretation of P and the ground atomic subset of L is called the Herbrand Base of P. I is called a Herbrand model of Horn theory P when I is both Herbrand and a model of P. According to Herbrand's theorem P is satisfiable iff it has a Herbrand model. Let F and G be two wffs. We say that F entails G, or $F \models G$, iff every model of F is a model of G.

Proof for LPs

An inference rule $I = F \rightarrow G$ states that wff F can be rewritten by wff G. We say $F \vdash_I G$ iff there exists a series of applications of I which transform F to G. I is said to be *sound* iff for each $F \vdash_I G$ always implies $F \models G$ and *complete* when $F \models G$ always implies $F \vdash_I G$. I is said to be *refutation complete* if I is complete with G restricted to False. The substitution θ is said to be the unifier of the atoms a and a' whenever $a\theta = a'\theta$. μ is the most general unifier (mgu) of a and a' if and only if for all unifiers γ of a and a' there exists a substitution δ such that $(a\mu)\delta = a\gamma$. The resolution inference rule is as follows. $((C \setminus \{a\}) \cup (D \setminus \{\neg a'\}))\theta$ is said to be the resolvent of the clauses C and D whenever C and D have no common variables, $a \in C$, $\neg a' \in D$ and θ is the mgu of a and a'. Suppose P is a definite program and G is a goal. Resolution is linear when D is restricted to clauses in P and C is either G or the resolvent of another linear resolution. The resolvent of such a linear resolution is another goal. Assuming the literals in clauses are ordered, a linear resolution is SLD when the literal chosen to resolve on is the first in C. An SLD refutation from P is a sequence of such SLD linear resolutions, which can be represented by $D_{P,G} = \langle G, C_1, .., C_n \rangle$ where each C_i is in P and the last resolvent is the empty clause (ie. False). The answer substitution is $\theta_{P,G} = \theta_1\theta_2..\theta_n$ where each θ_i is the substitution corresponding with the resolution involving C_i in $D_{P,G}$. If P is range-restricted then $\theta_{P,G}$ will be ground. SLD resolution is known to be both sound and refutation complete for definite programs. Thus for a range-restricted definite program P and ground atom a it can be shown that $P \models a$ by showing that $P, \leftarrow a \vdash_{SLD}$ False. The Negation-by-Failure (NF) inference rule says that $P, \leftarrow a \not\vdash_{SLD}$ False implies $P \vdash_{SLDNF} \neg a$.

SLPs

Syntax of SLPs

An SLP S is a set of labelled clauses $p{:}C$ where p is a probability (ie. a number in the range $[0, 1]$) and C is a first-order range-restricted definite clause[1]. The subset S_p of clauses in S with predicate symbol p in the head is called the definition of p. For each definition S_p the sum of probability labels π_p must be at most 1. S is said to be complete if $\pi_p = 1$ for each p and incomplete otherwise. $P(S)$ represents the definite program consisting of all the clauses in S, with labels removed.

Example 1 Unbiased coin. *The following SLP is complete and represents a coin which comes up either heads or tails with probability 0.5.*

$$S_1 = \left\{ \begin{array}{l} 0.5 : coin(head) \leftarrow \\ 0.5 : coin(tail) \leftarrow \end{array} \right\}$$

S_1 is a simple example of a sampling distribution[2].

[1] Cussens (Cussens 1999) considers a less restricted definition of SLPs.

[2] Section provides a more complex sampling distribution a language by attaching probability labels to productions of a grammar. The grammar is encoded as a range-restricted definite program.

المنارة للاستشارات

www.manaraa.com

Example 2 Pet example. *The following SLP is incomplete.*

$$S_2 = \left\{ \begin{array}{ll} 0.3 : likes(X,Y) \leftarrow & pet(Y,X), pet(Z,X), \\ & cat(Y), mouse(Z) \end{array} \right\}$$

S_2 shows how statements of the form $\Pr(P(\vec{x})|Q(\vec{y})) = p$ can be encoded within an SLP, in this case $\Pr(\text{likes(X,Y)}|\ldots)) = 0.3$.

Proof for SLPs

A Stochastic SLD (SSLD) refutation is a sequence $D_{S,G} = \langle 1{:}G, p_1{:}C_1, .., p_n{:}C_n\rangle$ in which G is a goal, each $p_i{:}C_i \in S$ and $D_{P(S),G} = \langle G, C_1, .., C_n\rangle$ is an SLD refutation from $P(S)$. SSLD refutation represents the repeated application of the SSLD inference rule. This takes a goal $p{:}G$ and a labelled clause $q{:}C$ and produces the labelled goal $pq{:}R$, where R is the SLD resolvent of G and C. The answer probability of $D_{S,G}$ is $Q(D_{S,G}) = \prod_{i=1}^{n} p_i$. The incomplete probability of any ground atom a with respect to S is $Q(a|S) = \sum_{D_{S,(\leftarrow a)}} Q(D_{S,(\leftarrow a)})$. We can state this as $S \vdash_{SSLD} Q(a|S) \leq Pr(a|S) \leq 1$, where $Pr(a|S)$ represents the conditional probability of a given S.

Remark 3 Incomplete probabilities. *If a is a ground atom with predicate symbol p and the definition S_p in SLP S is incomplete then $Q(a|S) \leq \pi_p$.*
Proof. *Suppose the probability labels on clauses in S_p are $p_1, .., p_n$ then $Q(a|S) = p_1q_1 + .. + p_nq_n$ where each q_i is a sum of products for which $0 \leq q_i \leq 1$. Thus $Q(a|S) \leq p_1 + .. + p_n = \pi_p$.*

Semantics of SLPs

In this section we introduce the "normal" semantics of SLPs. Suppose L is a first-order language and D_p is a probability distribution over the ground atoms of p in L. If I is a vector consisting of one such D_p for every p in L then I is called a distributional L-interpretation (or simply interpretation). If $a \in L$ is an atom with predicate symbol p and I is an interpretation then $I(a)$ is the probability of a according to D_p in I. Suppose L is chosen to be the smallest first-order language involving at least one constant and the predicate and function symbols of Horn theory $P(S)$. In this case an interpretation is called a distributional Herbrand interpretation of S (or simply Herbrand interpretation).

Definition 4 *An interpretation M is a distributional L-model (or simply model) of SLP S iff $Q(a|S) \leq M(a)$ for each ground atom a in L*[3].

Again if M is a model of S and M is Herbrand with respect to S then M is a distributional Herbrand model of S (or simply Herbrand model).

[3]It might seem unreasonable to define semantics in terms of proofs in this way. However, it should be noted that $Q(a|S)$ represents a potentially infinite summation of the probabilities of individual SSLD derivations. This is analogous to defining the satisfiability of a first-order formula in terms of an infinite boolean expression derived from truth tables of the connectives

Example 5 Models.

$$S = \left\{ \begin{array}{l} 0.5{:}p(X) \leftarrow q(X) \\ 0.5{:}q(a) \leftarrow \end{array} \right\}$$

$Q(p(a)|S) = 0.25$ and $Q(q(a)|S) = 0.5$. L has predicate symbols p, q and constant a, b.

$$I_1 = \left\langle \begin{array}{l} \{1{:}p(a), 0{:}p(b)\} \\ \{1{:}q(a), 0{:}q(b)\} \end{array} \right\rangle$$

I_1 is a model of S.

$$I_2 = \left\langle \begin{array}{l} \{0.1{:}p(a), 0.9{:}p(b)\} \\ \{0.5{:}q(a), 0.5{:}q(b)\} \end{array} \right\rangle$$

I_2 is not a model of S.

Suppose S, T are SLPs. As usual we write $S \models T$ iff every model of S is a model of T.

Learning SLPs

Bayes' function

This section describes a framework for learning a complete SLP S from examples E based on maximising Bayesian posterior probability $p(S|E)$. Below it is assumed that E consists of ground unit clauses. The posterior probability of S given E can be expressed using Bayes' theorem as follows.

$$p(S|E) = \frac{p(S)p(E|S)}{p(E)} \quad (1)$$

$p(S)$ represents a prior probability distribution over SLPs. If we suppose (as is normal) that the e_i are chosen randomly and independently from some distribution D over the instance space X then $p(E|S) = \prod_{i=1}^{m} p(e_i|S)$. We assume that $p(e_i|S) = Q(e_i|S)$ (see Section). $p(E)$ is a normalising constant. Since the probabilities involved in the Bayes' function tend to be small it makes sense to re-express Equation 1 in information-theoretic terms by applying a negative log transformation as follows.

$$-log_2p(S|E) = -log_2p(S) - \sum_{i=1}^{m}[log_2p(e_i|S)] + c \quad (2)$$

Here $-log_2p(S)$ can be viewed as expressing the size (number of bits) of S. The quantity $-\sum_{i=1}^{m}[log_2p(e_i|S)]$ can be viewed as the sum of sizes (number of bits) of the derivations of each e_i from S. c is a constant representing $log_2p(E)$. Note that this approach is similar to that described in (Muggleton 2000), differing only in the definition of $p(e_i|S)$. The approach in (Muggleton 2000) uses $p(e_i|S)$ to favour LP hypotheses with low generality, while Equation 2 favours SLP hypotheses with a low mean derivation size. Surprisingly this makes the Bayes' function for learning SLPs appropriate for finding LPs which have low time-complexity with respect to the examples. For instance, this function would prefer an SLP whose underlying LP represented quick-sort over one whose underlying LP represented insertion-sort since the mean proof lengths of the former would be lower than those of the latter.

المنارة للاستشارات
www.manaraa.com

Search strategy

The previous subsection leaves open the question of how hypotheses are to be constructed and how search is to be ordered. The approach taken in this paper involves two stages.

1. **LP construction.** Choose an SLP S with uniform probability labels on each definition and near maximal posterior probability with respect to E.
2. **Parameter estimation.** Vary the labels on S to increase the posterior probability with respect to E.

Progol4.5 is used to implement the search in Stage 1. Stage 2 is implemented using an algorithm which assigns a label to each clause C in S according to the Laplace corrected relative frequency with which C is involved in proofs of the positive examples in E.

Limitations of strategy

The overall strategy is sub-optimal in the following ways: a) the implementation of Stage 1 is approximate since it involves a greedy clause-by-clause construction of the SLPs, b) the implementation of Stage 2 is only optimal in the case that each positive example has a unique derivation.

Overview of Progol

ILP systems take LPs representing background knowledge B and examples E and attempt to find the simplest consistent hypothesis H such that the following holds.

$$B \wedge H \models E \tag{3}$$

This section briefly describes the Mode Directed Inverse Entailment (MDIE) approach used in Progol (Muggleton 1995). Equation 3 is equivalent for all B, H and E to the following.

$$B \wedge \overline{E} \models \overline{H}$$

Assuming that $\overline{H}$ and $\overline{E}$ are ground and that $\overline{\bot}$ is the conjunction of ground literals which are true in all models of $B \wedge \overline{E}$ we have the following.

$$B \wedge \overline{E} \models \overline{\bot}$$

Since $\overline{H}$ is true in every model of $B \wedge \overline{E}$ it must contain a subset of the ground literals in $\overline{\bot}$. Hence

$$B \wedge \overline{E} \models \overline{\bot} \models \overline{H}$$

and so for all H

$$H \models \bot \tag{4}$$

The set of solutions for H considered by Progol is restricted in a number of ways. Firstly, $\bot$ is assumed to contain only one positive literal and a finite number of negative literals. The set of negative literals in $\bot$ is determined by mode declarations (statements concerning the input/output nature of predicate arguments and their types) and user-defined restrictions on the depths of variable chains.

Progol uses a covering algorithm which repeatedly chooses an example e, forms an associated clause $\bot$ and searches for the clause which maximises the information compression within the following bounded sub-lattice.

$$\Box \preceq H \preceq \bot$$

The hypothesised clause H is then added to the clause base and the examples covered by H are removed. The algorithm terminates when all examples have been covered. In the original version of Progol (CProgol4.1) (Muggleton 1995) the search for each clause H involves maximising the 'compression' function

$$f = (p - (c + n + h))$$

where p and n are the number of positive and negative examples covered by H, c is the number of literals in H, and h is the minimum number of additional literals required to complete the input/output variable chains in H (computed by considering variable chains in $\bot$). In later versions of Progol the following function was used instead to reduce the degree of greediness in the search.

$$f = \frac{m}{p}(p - (c + n + h)) \tag{5}$$

This function estimates the overall global compression expected of the final hypothesised set of clauses, extrapolated from local coverage and size properties of the clause under construction. A hypothesised clause H is pruned, together with all its more specific refinements, if either

$$1 - \frac{c}{p} \leq 0 \tag{6}$$

or there exists a previously evaluated clause H' such that H' is an acceptable solution (covers below the noise threshold of negative examples and the input/output variable chains are complete) and

$$1 - \frac{c}{p} \leq 1 - \frac{c' + n' + h'}{p'} \tag{7}$$

where p, c are associated with H and p', n', c', h' are associated with H'.

User-defined evaluation in Progol4.5

User-defined evaluation functions in Progol4.5 are implemented by allowing redefinition in Prolog of p, n and c from Equation 5. Figure 1 shows the convention for names used in Progol4.5 for the built-in and user-defined functions for these variables. Though this approach to allowing definition of the evaluation function is indirect, it means that the general criteria used in Progol for pruning the search (see Inequalities 6 and 7) can be applied unaltered as long as user_pos_cover and user_neg_cover monotonically decrease and user_hyp_size monotonically increases with downward refinement (addition of body literals) to the hypothesised clause. For learning SLPs these functions are derived below.

المنارة للاستشارات
www.manaraa.com

Variable	Built-in	User-defined
p	pos_cover(P1)	user_pos_cover(P2)
n	neg_cover(N1)	user_neg_cover(N2)
c	hyp_size(C1)	user_hyp_size(C2)
h	hyp_rem(H1)	user_hyp_rem(H2)

Figure 1: Built-in and user defined predicates for some of the variables from Equation 5.

Equation 2 can be rewritten in terms of an information function I as

$$I(S|E) = I(S) - \sum_{i=1}^{m} I(e_i|S) + c \qquad (8)$$

where $I(x) = -\log_2 x$. The degree of compression achieved by an hypothesis is computed by subtracting $I(S|E)$ from $I(S' = E|E)$, the posterior information of the hypothesis consisting of returning ungeneralised examples.

$$\begin{aligned} I(S' = E|E) &= I(E) + I(E|S' = E) + c \\ &= m + m\log_2 m + c \\ &= m(1 + \log_2 m) + c \qquad (9) \end{aligned}$$

The compression induced by S with respect to E is now simply the difference between Equations 9 and 8, which is as follows.

$$\begin{aligned} & m(1+\log_2 m) - I(S) + \sum_{i=1}^{m} I(e_i|S) \\ = \; & \frac{m}{p}(p(1+\log_2 m) - I(H) + \sum_{j=1}^{p} I(e_j|H)) \qquad (10) \end{aligned}$$

In Equation 10 extrapolation is made from the p positive examples covered by hypothesised clause H. Comparing Equations 5 and 10 the user-defined functions of Figure 1 are as follows (p, n, c, h represent built-in functions and p', n', c', h' represent their user-defined counter-parts).

$$\begin{aligned} p' &= p(1+\log_2 m) \qquad (11) \\ n' &= \sum_{j=1}^{m} I(e_j|H) + n \\ c' &= c \\ h' &= h \end{aligned}$$

Worked examples

The source code of Progol4.5 together with the input files for the following worked examples can be obtained from ftp://ftp.cs.york.ac.uk/pub/mlg/progol4.5/ .

Animal taxonomy

Figure 2 shows the examples and background knowledge for an example set which involves learning taxonomic descriptions of animals. Following Stage 1 (Section) the SLP constructed has uniform probability labels as follows[4].

Examples	class(dog,mammal). class(trout,fish). ...
Background knowledge	has_covering(dog,hair). ... has_legs(dolphin,0). ...

Figure 2: Examples and background knowledge for animal taxonomy.

```
0.200: class(A,reptile) :-
        has_legs(A,4), has_eggs(A).
0.200: class(A,mammal) :- has_milk(A).
0.200: class(A,fish) :- has_gills(A).
0.200: class(A,reptile) :-
        has_legs(A,0), habitat(A,land).
0.200: class(A,bird) :-
        has_covering(A,feathers).
```

Following Stage 2 the labels are altered as follows to reflect the distribution of class types within the training data.

```
0.238: class(A,reptile) :-
        has_legs(A,4), has_eggs(A).
0.238: class(A,mammal) :- has_milk(A).
0.238: class(A,fish) :- has_gills(A).
0.095: class(A,reptile) :-
        has_legs(A,0), habitat(A,land).
0.190: class(A,bird) :-
        has_covering(A,feathers).
```

Simple English grammar

Examples	s([the,man,walks,the,dog],[]). s([the,dog,walks,to,the,man],[]). ...
Background knowledge	np(S1,S2) :- det(S1,S3), noun(S3,S2). ... noun([man|S],S). ...

Figure 3: Examples and background knowledge for Simple English Grammar.

Figure 3 shows the examples and background knowledge for an example set which involves learning a simple English grammar. Following Stage 2 the learned SLP is as follows.

```
0.438: s(A,B) :- np(A,C), vp(C,D),
        np(D,B).
```

[4]For this example and the next the value of p' (Equation 11) was increased by a factor of 4 to achieve positive compression

المنارة للاستشارات
www.manaraa.com

```
0.562: s(A,B) :- np(A,C), verb(C,D),
       np(D,E), prep(E,F), np(F,B).
```

Conclusion

This paper describes a method for learning SLPs from examples and background knowledge. The method is based on an approximate Bayes MAP (Maximum A Posterior probability) algorithm. The implementation within Progol4.5 is efficient and produces meaningful solutions on simple domains. However, as pointed out in Section the method does not find optimal solutions.

The author views the method described as a first attempt at a hard problem. It is believed that improvements to the search strategy can be made. This is an interesting topic for further research.

The author believes that learning of SLPs is of potential interest in all domains in which ILP has had success (Muggleton 1999a). In these domains it is believed that SLPs would the advantage over LPs of producing predictions with attached degrees of certainty. In the case of multiple predictions, the probability labels would allow for relative ranking. This is of particular importance for Natural Language domains, though would also have general application in Bioinformatics (Muggleton 1999b).

Acknowledgements

The author would like to thank Wray Buntine, David Page, Koichi Furukawa and James Cussens for discussions on the topic of Stochastic Logic Programming. Many thanks are due to my wife, Thirza and daughter Clare for the support and happiness they give me. This work was supported partly by the Esprit RTD project "ALADIN' (project 28623), EPSRC grant "Closed Loop Machine Learning", BBSRC/EPSRC grant "Protein structure prediction - development and benchmarking of machine learning algorithms" and EPSRC ROPA grant "Machine Learning of Natural Language in a Computational Logic Framework".

References

Cussens, J. 1999. Loglinear models for first-order probabilistic reasoning. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence*, 126–133. San Francisco: Kaufmann.

Fagin, R., and Halpern, J. 1989. Uncertainty, belief and probability. In *Proceedings of IJCAI-89*. San Mateo, CA: Morgan Kauffman.

Jaeger, M. 1997. Relational bayesian networks. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA: Kaufmann.

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

Lloyd, J. 1987. *Foundations of Logic Programming*. Berlin: Springer-Verlag. Second edition.

Muggleton, S. 1995. Inverse entailment and Progol. *New Generation Computing* 13:245–286.

Muggleton, S. 1996. Stochastic logic programs. In de Raedt, L., ed., *Advances in Inductive Logic Programming*. IOS Press. 254–264.

Muggleton, S. 1999a. Inductive logic programming: issues, results and the LLL challenge. *Artificial Intelligence* 114(1–2):283–296.

Muggleton, S. 1999b. Scientific knowledge discovery using inductive logic programming. *Communications of the ACM* 42(11):42–46.

Muggleton, S. 2000. Learning from positive data. *Machine Learning*. Accepted subject to revision.

المنارة للاستشارات

www.manaraa.com