Two Novel Methods for Learning Logic Programs with Annotated Disjunctions

Hendrik Blockeel

Katholieke Universiteit Leuven, Department of Computer Science
Celestijnenlaan 200A, 3001 Leuven, Belgium

Abstract. Logic programs with annotated disjunctions (LPAD’s) are an elegant formalism for combining first order logic and probabilistic reasoning. Methods for learning restricted classes of LPAD’s have been proposed before. In this paper two approaches for learning broader classes of LPAD’s are proposed. One uses a reduction to bayesian networks, allowing us to exploit the many learning methods developed for those, but leaving some restrictions on the LPAD’s that can be learned. The second works for any LPAD, but its scalability to large LPAD’s is currently an open issue.

1 Introduction

Logic programs with annotated disjunctions (LPAD’s) have been introduced by Vennekens et al. [1] as a knowledge representation formalism that allows to combine probabilistic and logic inference. In essence, an LPAD rule is a first order logic clause where the head literals are annotated with probabilities. It is usually written \((h_1: \alpha_1), (h_2: \alpha_2), \ldots, (h_n: \alpha_n) \leftarrow b_1, b_2, \ldots, b_m\) where the \(h_i\) and \(b_i\) are literals and \(\alpha_i \in [0,1]\). In a procedural setting, one can interpret the probabilities as follows: when the rule fires, it makes exactly one of the literals in the head true, and for each head literal \(h_i\) the probability that it is selected is the corresponding \(\alpha_i\).

Note that a literal may occur in the head of different LPAD rules. When a literal occurs in only one head, \(\alpha_i\) can simply be interpreted as \(P(h_i|B)\) with \(B\) the body of the rule where \(h_i\) occurs in the head. But if, say, a literal \(h\) occurs in two rules with bodies \(B\) and \(C\), \(h\) may be true even if the rule with \(B\) did not select it but the one with \(C\) did. We then have \(P(h|B) = \alpha + (1 - \alpha)P(B \land C)\beta\) with \(\alpha\) and \(\beta\) the respective annotations of \(h\) in the rules with bodies \(B\) and \(C\). In general, \(\alpha_i\) is to be interpreted as \(P(h_i|B, \neg X)\) where \(X\) is the disjunction of all other conditions that might make \(h_i\) true.

2 Learning LPAD’s

Riguzzi [2] proposed a method for learning LPAD’s under the restriction that literals can only occur in the head of multiple clauses if the bodies of these clauses are mutually exclusive. We call this the ME-restriction. Under that condition,
\( \alpha_i = P(h_i | B) \) for all \( h_i : \alpha_i \) in a rule with body \( B \). Such \( \alpha \)'s can be learned easily, since \( P(h_i | B) \) can be estimated as \( |M(B)|/|M(B \land h_i)| \) where \( M(F) \) is the set of examples (interpretations) for which formula \( F \) holds.

But the ME-restriction prohibits certain kinds of LPAD’s that might be interesting to consider. For instance, the LPAD

\[
\begin{align*}
\text{hairwet} : 0.5 & \leftarrow \text{rain} \\
\text{hairwet} : 0.7 & \leftarrow \text{showered}
\end{align*}
\]

expresses that if someone takes a shower, their hair is wet afterwards with a probability of 0.7, and walking in the rain causes it to be wet with a probability of 0.5. These probabilities are very concretely interpretable, yet none of these probabilities is equal to “the conditional probability of someone’s hair being wet after showering” (or after walking in the rain). Rather, the \( \alpha \)'s reflect the probability that the body \textit{causes} the head, or: the probability that the head is true given the body and assuming \textit{nothing else has already made the head true}.

The problem with learning such LPAD’s is that the conditional probabilities can be estimated easily from data, but the \( \alpha_i \) cannot. In the following we discuss two ways in which the \( \alpha_i \)'s could be obtained. The first is through a connection with Bayesian nets; it gives us some insight in how the \( \alpha_i \) and the \( P(h_i | B) \) relate. The second is a more direct approach.

### 2.1 Learning LPAD’s through Bayesian nets

Consider the following definitions. An LPAD is \textbf{ME-compliant} if for any literals that occur in the head of multiple rules, the bodies are mutually exclusive. An LPAD is \textbf{1-compliant} if for each \( h_i : \alpha_i \) in the head of an LPAD rule it holds that \( h_i \) occurs in no other rules or \( \alpha_i = 1 \). An LPAD is \textbf{CP-compliant} if for each \( h_i : \alpha_i \) occurring in rule with body \( B \) it holds that \( \alpha_i = P(h_i | B) \).

We then have the following properties. \textbf{ME-compliant LPAD’s are CP-compliant}. (This is the property that Riguzzi exploits in his approach.) \textbf{1-compliant LPAD’s are CP-compliant} (proof sketch: trivial if \( h_i \) occurs in only one head; if it occurs more, then \( P(h_i | B) \geq \alpha_i = 1 \) hence \( P(h_i | B) = 1 \)).

\textbf{Any LPAD can be converted into a 1-compliant LPAD}: introduce for each occurrence of \( h_i \) a unique predicate symbol, and introduce separate Horn clauses defining \( h_i \) in terms of the unique predicates. For the above example, this would give:

\[
\begin{align*}
\text{hairwet}_1 : 0.5 & \leftarrow \text{rain} \\
\text{hairwet}_2 : 0.7 & \leftarrow \text{showered} \\
\text{hairwet} & \leftarrow \text{hairwet}_1 \\
\text{hairwet} & \leftarrow \text{hairwet}_2
\end{align*}
\]

This transformation makes the program 1-compliant, hence CP-compliant, which means the \( \alpha_i \) can now be estimated easily, except that the newly introduced predicate symbols do not occur in our original alphabet and hence we do not know in how many examples, e.g., \texttt{hairwet\_i} is true. This situation is similar to
learning the parameters of a graphical model (a Bayesian net, for instance) with unobserved variables. A non-recursive LPAD can be transformed into a Bayesian network [1]. Hence, any non-recursive LPAD can be learned by transforming it into its 1-compliant equivalent, transforming that into a Bayesian net (which will have some unobserved variables), then learning the Bayesian net from the available data using EM or one of the other methods available for this.

This approach removes the restriction that only ME-compliant LPAD’s can be learned, but still does not allow recursive LPAD’s to be learned. Yet it has the advantage that standard techniques from learning Bayesian nets can be used.

2.2 Learning LPAD’s directly

The semantics of an LPAD is defined as a probability distribution over interpretations [1], which in turn defines a probability distribution over all logical formulae. The probability of any logical formula can thus be written as a function of the α’s, and it can of course also be estimated from the data. One can therefore estimate the α’s as follows: take some set of logical formulae, write the probability of each formula as a function of the α’s and also estimate it from the data. This gives a system of equations that can be solved for the α’s.

A prototype implementation of the above approach was made by the author. Given a set of conjunctions with corresponding probabilities, and an LPAD with symbolic α’s, it constructs formulae expressing the probability of each conjunction in terms of the α’s and then tries to solve the system for the α’s. A wrapper around this allows structure learning. The method works well on small toy programs but has not been evaluated yet for larger LPAD’s.

3 Conclusion

It is unnecessary to restrict the learning of LPAD’s to specific subclasses. Non-recursive LPAD’s can be learned using standard BN learning techniques using the reduction proposed in this paper, and any LPAD can in principle be learned using the direct learning technique.

Acknowledgements

Hendrik Blockeel is a post-doctoral fellow of the Research Foundation of Flanders (FWO-Vlaanderen).

References