An Abductive-Inductive Algorithm for Probabilistic Inductive Logic Programming

Stanislav Dragiev, Alessandra Russo, Krysia Broda, Mark Law, and Rares Turliuc

Department of Computing, Imperial College London, United Kingdom
{sd4312, a.russo, kb, ml1909, calin-rares.turliuc}@imperial.ac.uk

Abstract. The integration of abduction and induction has lead to a variety of non-monotonic ILP systems. `XHAIL` is one of these systems, in which abduction is used to compute hypotheses that subsume Kernel Sets. On the other hand, `Peircebayes` is a recently proposed logic-based probabilistic programming approach that combines abduction with parameter learning to learn distributions of most likely explanations. In this paper, we propose an approach for integrating probabilistic inference with ILP. The basic idea is to redefine the inductive task of `XHAIL` as a statistical abduction, and to use `Peircebayes` to learn probability distribution of hypotheses. An initial evaluation of the proposed algorithm is given using synthetic data.

1 Introduction

Inductive Logic Programming (ILP) is concerned with learning logic programs that, together with a background knowledge, explain given examples (or data). One of the greatest benefits of ILP, with respect to other not logic-based machine learning, is that the learned programs are very easy to interpret in natural language. A disadvantage is, however, that it does not handle noise and uncertainty in the observations very well. On the other hand, machine learning methods that relay on Bayesian statistics are better in coping with uncertainty. Their objective is to find parameters of statistical models that maximize the likelihood of the data. Inference on these models can be very efficient when applied to large data sets, but coming up with a correct probabilistic model, which is able to capture the conditional dependencies of the data may turn up to be a difficult task. Combining logic-based learning (or logic-based inference in general) and Bayesian statistics may result in symbolic models that can cope with uncertainty, benefiting from both higher accuracy and easy interpretation.

In [9] a probabilistic abductive logic programming approach has been proposed, called `Peircebayes`, which combines abductive inference and parameter learning through Bayesian prior and sampling. The underlying probabilistic model involves categorical variables with conjugate Dirichlet priors and it is represented as an abductive logic program where abducibles correspond to the probabilistic categorical variables of the model. The program is augmented with a notion of *plate*, which essentially executes an abductive query for each given observation and generates as abductive solutions boolean formulae over the

abducibles that express the categorical variables relevant to explain the given observations. An adaptation of the Gibbs sampling algorithm [4] is used to learn the probability distribution over the categorical variables.

Abduction has also been shown to be useful in inductive learning. The `XHAIL` approach [7] integrates abductive and inductive inference to compute solutions for non-monotonic inductive learning tasks. The algorithm generates first a (ground) most specific Kernel Set (i.e. a set of ground normal clauses) that, together with the background knowledge, explain the given set of examples, and then uses abduction to solves the inductive step of searching for the most general hypothesis (set of clauses) that subsumes the Kernel set. This requires a special rewriting of the Kernel Set (as summarised in Section 2). `XHAIL` is however limited in handling noisy data.

This paper explores a way of combining probabilistic inference with ILP in order to learn not only structured hypothesis but also probabilisitic distributions over the clauses that form such hypothesis. The method consists of re-defining the inductive step of the `XHAIL` algorithm with a statical abduction computation. The Kernel Set, derived from the language bias of a given learning task, is redefined as a special type of annotated literal program, in which each literal is associated with a unique probabilistic parameter. Such a re-written Kernel is then taken as the input statistical abductive program of `Peircebayes`. It is used as generative story for the computation of literals in the hypothesis that are (logically) relevant to the given data, and statistically for the learning of probability distributions over these literals. The outcome is the learning of probabilistic distributions over clauses that form inductive solutions. Preliminary evaluation over synthetic data show some promising results.

The paper is organised as follows. First, we cover some background on `XHAIL` and `Peircebayes`. In Section 3, we illustrate our new `PROBXHAIL` algorithm with an example. In Section 4, we define the semantics of the Annotated Literal Programs, a new kind of probabilistic logic program, which are output by `PROBXHAIL`. We conclude with a brief evaluation and final remarks.

2 Background

In this section, we briefly introduce `XHAIL` and `Peircebayes`, the two building blocks of our proposed approach.

XHAIL

`XHAIL` is an ILP algorithm that combines induction and abduction in several ways. It uses an abductive-deductive procedure to compute a Kernel Set, a set of (normal) clauses that represent a most-specific hypothesis that cover the given data. Then, it performs induction on the Kernel Set to compute a most general hypothesis that subsumes the Kernel Set, whilst preserving the same coverage. This step is re-expressed as an abductive task in which each explanation represents a single hypothesis that subsumes the Kernel Set and does not contradict any example. This inductive procedure is of particular interest for this paper. We

show how to replace this step with statical abduction in order to perform Probabilistic ILP. In what follows we present the abductive representation used by XHAIL for computing inductive solutions and the statistical abduction for computing a probability distribution on abducibles. In Section 3 we describe how our approach combines these two techniques. Consider a set of $C = \{1 \dots k\}$ clauses of the form $\alpha_c \leftarrow \delta_c^1, \dots, \delta_c^{m_c}$. The associated abductive logic programming *reformulation* of such a set is the program Π obtained by replacing every clause $\alpha_c \leftarrow \{\delta_c^i\}_{i=1}^{m_c}$ by the following set of clauses:

- $\alpha_c \leftarrow use(c,0), try(c,1,vars(\delta_c^1)), \dots, try(c,m_c,vars(\delta_c^{m_c}))$, where $vars(\delta)$ represents the list of variables in the literal δ.
- for every body literal in $\{\delta_c^i\}_{i=1}^{m_c}$ the following two clauses are included: $try(c,i,vars(\delta_c^i)) \leftarrow not\ use(c,i).\ try(c,i,vars(\delta_c^i)) \leftarrow \delta_c^i, use(c,i).$

In the above reformulation, the ground literals $use(c,i)$ and $not\ use(c,i)$ constitute the set of abducibles.

Example 1. Let us assume K to be the most specific Kernel Set computed by XHAIL for a given set E of examples. The inductive generalisation step of XHAIL constructs the associated abductive task $< \Pi, A, E >$, where Π and A are defined as follows[1]:

$$K = \left\{ \begin{array}{l} a \leftarrow b. \\ b. \end{array} \right\} \qquad \Pi = \left\{ \begin{array}{l} a \leftarrow use(0,0), try(0,1). \\ try(0,1) \leftarrow b, use(0,1). \\ try(0,1) \leftarrow not\ use(0,1). \\ b \leftarrow use(1,0). \end{array} \right\} \quad \mathcal{A} = \left\{ \begin{array}{l} use(0,0). \\ use(0,1). \\ use(1,0). \end{array} \right\}$$

Peircebayes

Traditionally, abduction is the task of infering a set of ground atoms, called an explanation, that together with a given background theory entails a specific observation given by a conjunction of (ground) facts. This concept can be generalized to probabilistic abduction by introducing probability distributions over the truth values of each (ground) abducible and over the observations. This probabilistic perspective provides a method of quantitatively estimating the quality of the abductive solutions. Introducing probability in abduction essentially redefines the notion of abductive solutions as no longer the minimal but the most preferred (possibly non minimal) assumptions, based on their probability, needed to explain a specific distribution of observations (e.g., [6], [3]).

Peircebayes is an algorithm that attempts to perform inference in a specific probabilistic model for statistical abduction. In it, each abducible is associated with an independent categorical variable θ (referred to as *parameter*) with a user-specified Dirichlet prior α (referred to as *hyperparameter*). The inference task is to compute the probability distribution of the parameters given the observations.

[1] Note that in the actual implementation of the XHAIL algorithm, the abductive program also uses type matching for the bias and to ensure safe rules.

In probabilistic terms the observations can be seen as a vector f of N data points, where $f_n = 1$ denotes that the n-th observation is included in the model. The novelty is that the draw of the values for the parameters θ is related to the "generative story" of the model expressed as an abductive task.d

In `Peircebayes` the inference task is to determine a distribution of most likely explanations, i.e. to infer $P(\theta|f;\alpha)$. Since the prior distribution is conveniently chosen to be a Dirichlet distribution, the prior conjugate of the categorical distribution, this task is equivalent to finding posterior values of α. The posterior expectation of θ can be computed via uncollapsed Gibbs Sampling along $P(x|\theta, f)$ and $P(\theta|x, \alpha)$. Sampling from $P(\theta|x, \alpha)$ involves sampling from a Dirichlet distribution (since the posterior of a categorical distribution with a Dirichlet prior is also a Dirichlet distribution). Sampling from $P(x|\theta, f)$ can be done by sampling paths in BDDs that describe the category of abducibles in the abductive solutions of each observation. The reader is referred to [9] for further details.

3 PROBXHAIL

This section describes the `PROBXHAIL` algorithm. For simplicity, we illustrate its workings with a running example with propositional atoms. The program takes as input a set of (partial) interpretations (see [5]) and a bias.

Example 2. `PROBXHAIL` could consist of the following input. The example set over 100000 observations could consist of $\langle\{a, b\}, \{\}\rangle$ repeated 62958 times, $\langle\{a\}, \{b\}\rangle$ repeated 2727 times, $\langle\{b\}, \{a\}\rangle$ repeated 27109 times and $\langle\{\}, \{a, b\}\rangle$ repeated 7206 times. Assume a mode bias that consists of `modeh(a)`, `modeb(b)`, `modeh(b)`, `determination(a,b)` and prohibits repetition of clauses.

The output of `PROBXHAIL` is a probabilistic logic program, a normal logic program augmented with statistical parameters. Such a program has non-deterministic semantics: it defines a distribution over deterministic normal logic programs and, hence, over partial interpretations of the program. For this reason, the number of occurrences of each example is relevant: the example set represents an empirical distribution of partial interpretations and the goal of `PROBXHAIL` is to produce a probabilistic logic program that defines the same probability distribution over the examples. We describe the semantics of the probabilistic logic programming language in Section 4.

Overall, the algorithm consists of two phases: structure learning and parameter learning. During the structure learning phase, a deterministic normal logic program is constructed using the bias. During the parameter learning phase, the parameters for a set of Bernoulli random variables is learned, one for every literal. The parameters are such that the probability distribution of partial interpretations defined by the program is as close as possible to the probability distribution implied by the examples.

More specifically, the structural learning phase of the algorithm is currently purely bias-driven. During this step, the most specific logic program that agrees with the bias is constructed. The bias is similar to the one defined in other

ILP systems and includes mode and determination declarations. In addition, it specifies the number of times a clause can be repeated (this is relevant in the probabilistic logic program, unlike in deterministic programs where clause repetitions are redundant). With propositional programs, this procedure involves constructing a clause for every head literal and then adding every body literal that agrees with the determinations.

This procedure can be generalized to normal logic programs with variable terms (as shown in [2]). Furthermore, an alternative data-driven structural learning approach could be used which is similar to `XHAIL` and generates a smaller logic theory. This results in faster probabilistic inference.

In the next step, parameter learning is performed. This is done by performing `XHAIL`s inductive task transformation on the theory generated from the structural learning step and then performing `Peircebayes` statistical abduction with the generated background and the `use` ground atoms as abducibles and the input examples as observations. This produces an unbiased estimate of the parameters of the probabilistic logic program.

Example 3. Structural learning for Example 2 yields $\{a \leftarrow b.\ b.\}$. We construct the following `Peircebayes` task with observations $\mathcal{O}$ from Example 2 and background B and abducibles A, given in Example 1. The output of the `PROBXHAIL` algorithm is a probabilistic program in which every literal is associated with a Bernoulli trial parameter:

$$P = \left\{ \begin{array}{l} a : 0.699 \leftarrow b : 604. \\ b : 0.9. \end{array} \right\}$$

It can be shown that the learned parameters minimize the posterior expected value of the root mean square error function of θ.

4 Annotated Literal Programs

In this section, we define *Annotated Literal Programs* (the output of `PROBXHAIL`). In these programs, every literal is associated with a probabilistic Bernoulli parameter. Annotated literal programs define a probability distribution on normal logic programs. Informally, we can define this probability distribution by the method we use to sample: every clause is considered independently and is included with the probability of its head literal. Only a subset of the body literals is retained in the clause. Each body literal is considered independently and included in the clause with its associated probability. Annotated literal programs are based on answer sets semantics: the probability of a query being successful is the sum of the probability of all normal logic programs that bravely entail it (i.e. the query is true in at least one of its answer sets).

Example 4. The probability distribution on logic programs described by the program in Example 3 is shown in the table below. The probability of query `?-a.` is the sum of the probability of the normal logic programs that entail it. In this example, T_1, T_4 and T_5 have answer sets which satisfy the query. Thus, the probability of the program is $p(T_1) + p(T_4) + p(T_5) = 0.6567804$.

	Logic Program	Answer Set	Probability
T_1	$\{a \leftarrow b.\ b.\}$	$\{a.b.\}$	$0.699 \times 0.604 \times 0.9 = 0.3799764$
T_2	$\{a \leftarrow b.\}$	$\emptyset$	$0.699 \times 0.604 \times (1 - 0.9) = 0.0422196$
T_3	$\{b.\}$	$\{b.\}$	$(1 - 0.699) \times 0.9 = 0.2709$
T_4	$\{a.\ b.\}$	$\{a.b.\}$	$0.699 \times (1 - 0.604) \times 0.9 = 0.2491236$
T_5	$\{a.\}$	$\{a.\}$	$0.699 \times (1 - 0.604) \times (1 - 0.9) = 0.0276804$
T_6	$\emptyset$	$\emptyset$	$(1 - 0.699) \times (1 - 0.9) = 0.0301$

Note that sampling from the probabilistic logic program involves a series of Bernoulli trials. During parameter learning in `PROBXHAIL` we can apply a Bayesian prior on the distributions in order to define a probabilistic bias. Currently only a symmetric Beta prior is used (as `Peircebayes` supports Dirichlet Priors on abducibles); however, the applications of a user-defined prior is an open question.

It can be shown via a translation procedure [2] that Annotated Literal Programs can represent ProbLog[1] programs and also other programs based on distribution semantics[8]. An important difference to note is that ProbLog is based on the SLDNF semantics, and Annotated Literal Programs is based on the answer set semantics.

5 Evaluation and Final Remarks

We run an experiment in order to measure parameter learning error. We sample normal logic programs multiple times from a given probabilistic logic program and then compute their full interpretations. We use the resulting interpretations as input for `PROBXHAIL` and attempt to reconstruct the original probabilistic logic program used for the sampling. We then measure the runtime of learning and the root mean square error in the parameters learned versus the original parameters. For instance, the observations in Example 2 were generated from the following probabilistic program:

$$P = \left\{ \begin{array}{l} a : 0.7 \leftarrow b : 0.6. \\ b : 0.9. \end{array} \right\}$$

Results show that the root mean square error of parameter learning in Example 3 decreases below 0.001 in 3 iterations of uncollapsed Gibbs sampling. Experiments with larger synthetic datasets suggest that the number of iterations necessary for convergence increases with the number of literals.

Overall, `PROBXHAIL` is prone to issues with its high-dimensional parametric search space. For example, the number of different possible observations increases exponentially with the number of clauses. Moreover, increasing the number of parameters requires an increased number of samples and `Peircebayes` iterations for convergence. As a consequence of this, maintaining the number of literals as low as possible is important. This implies that a structural learning procedure is necessary that minimizes the size of the probabilistic logic program.

References

1. Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2468–2473, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.
2. Stanislav Dragiev. PROBXHAIL: An abductive-inductive algorithm for probabilistic inductive logic programming. Master's thesis, Imperial College London, June 2016.
3. Daan Fierens, Guy Van den Broeck, Ingo Thon, Bernd Gutmann, and Luc De Raedt. Inference in probabilistic logic programs using weighted cnf's. In *Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pages 211–220, Corvallis, Oregon, 2011. AUAI Press.
4. Masakazu Ishihata and Taisuke Sato. Bayesian inference for statistical abduction using markov chain monte carlo. In *ACML*, pages 81–96, 2011.
5. Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs. In *Logics in Artificial Intelligence*, pages 311–325. Springer, 2014.
6. David Poole. Abducing through negation as failure: stable models within the independent choice logic. *J. Log. Program.*, 44(1-3):5–35, 2000.
7. Oliver Ray. Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, 7(3):329 – 340, 2009.
8. Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming*, pages 715–729. MIT Press, 1995.
9. Calin Rares Turliuc, Luke Dickens, Alessandra Russo, and Krysia Broda. Probabilistic abductive logic programming using dirichlet priors. *International Journal of Approximate Reasoning*, 2016.