Learnability of Constrained Logic Programs

Sašo Džeroski[1], Stephen Muggleton[2], Stuart Russell[3]

[1] Institut Jožef Stefan, Jamova 39, 61111 Ljubljana, Slovenia
[2] Oxford University Computing Laboratory, 11 Keble Road, Oxford OX1 3QD, UK
[3] Computer Science Division, University of California, Berkeley, CA94720, USA

Abstract. The field of *Inductive Logic Programming* (ILP) is concerned with inducing logic programs from examples in the presence of background knowledge. This paper defines the ILP problem and describes several syntactic restrictions that are often used in ILP. We then derive some positive results concerning the learnability of these restricted classes of logic programs, by reduction to a standard propositional learning problem. More specifically, k-literal predicate definitions consisting of constrained, function-free, nonrecursive program clauses are polynomially PAC-learnable under arbitrary distributions.

1 Introduction

The theory of Probably Approximately Correct (PAC) learning has been applied principally to propositional concept classes. Despite their successes, propositional learning approaches suffer from the limited expressiveness of their hypothesis language. Learning systems that use more expressive languages, have recently attracted a substantial amount of research in the machine learning community. As the learned hypothesis most often takes the form of a logic program, the field has been named Inductive Logic Programming (ILP) [10].

In this paper we will concentrate on the problem of learning a single concept or *target predicate*. Other work in ILP has focussed on learning several, possibly interdependent, concepts [14, 2]. Few PAC-learnability results have been established for either case (see Section 2), although the multiple-concept learning methods of [14] and [2] identify the correct concept in the limit.

Some ILP systems, such as LINUS [7], FOIL [12], and mFOIL [3], work by extending propositional approaches to a first-order framework. While FOIL and mFOIL use heuristic search techniques from propositional learning to construct first-order clauses directly, LINUS explicitly converts the first-order representation to a propositional one by defining an appropriate set of Boolean features.

In the paper, we use the transformation approach to obtain learnability results for logic programs by direct application of existing propositional PAC-learnability results. Section 2 defines the ILP problem and some syntactic restrictions, illustrating them in the context of a simple example. Section 3 shows how an ILP problem can be transformed to a propositional problem, and Section 4 gives the principal results. Section 5 discusses ways in which the imposed restrictions might be relaxed and suggests topics for further research.

2 Inductive Logic Programming

In the rest of this paper, we will assume that the reader is familiar with logic programming terminology as defined in the standard textbook of Lloyd [8]. In the logical framework we have adopted, a concept is a predicate. When expressed as a logic program, a concept (predicate) definition is a set of clauses, each having the same (target) predicate in its head.

The task of ILP is defined as follows. Given is a set $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$ of positive $\mathcal{E}^+$ and negative $\mathcal{E}^-$ examples of a target predicate, represented as ground literals. Background knowledge $\mathcal{B}$, a set of normal program clauses, typically including further facts describing the examples in $\mathcal{E}^+$ and $\mathcal{E}^-$, is also given. The task is to find a hypothesis $\mathcal{H}$, also a set of normal program clauses, such that $\mathcal{B} \wedge \mathcal{H} \models \mathcal{E}^+$ (where $\models$ is the relation of logical entailment), and $\forall e \in \mathcal{E}^- : \mathcal{B} \wedge \mathcal{H} \not\models e$. We define l to be the number of distinct predicates $p_1 \ldots p_l$ in $\mathcal{B}$, and n to be the arity (number of arguments) of the target predicate q.

In practice, ILP systems work within various syntactic restrictions in order to limit the complexity of the problem. In a *k-literal predicate definition* each clause has at most k literals in its body. A *k-clause predicate definition* consists of up to k clauses. A k-clause definition corresponds to a first-order k-term DNF formula and a k-literal definition to a first-order k-DNF formula.

Several additional restrictions are defined below. First of all, we assume an integer constant j is given. We consider only problems where all predicates in $\mathcal{B}$ are of arity not greater than j. We also require the background knowledge $\mathcal{B}$ to be efficient. We call $\mathcal{B}$ *efficient* if all atomic queries to it can be answered in time polynomial in the arity of the query predicate.

Several types of restrictions can be imposed on the clauses that form the hypothesis. A clause is *constrained* if all variables in its body also appear in the head. A clause is *function-free* if it mentions no function symbols. A clause is *nonrecursive* if the predicate symbol in its head does not appear in any of the literals in its body. A predicate definition is constrained/function-free/nonrecursive if all the clauses in it are constrained/function-free/nonrecursive. Although mutually recursive predicate definitions are not considered recursive above, this causes no problems as we are concerned with learning a definition of a single predicate, given definitions of some other predicates.

Let us illustrate the above definitions on the ILP problem of learning the *daughter* relationship. The task is to define the target relation $daughter(X, Y)$, which states that person X is a daughter of person Y, in terms of the background knowledge predicates *female* and *parent*, the latter being defined in terms of the relations *mother* and *father*. All these relations are given in Table 1. There are two positive and two negative examples of the target relation.

A 1-*clause* definition of the target predicate in terms of the specified background knowledge predicates is: $daughter(X, Y) \leftarrow female(X), parent(Y, X)$. In the above terminology, this hypothesis is nonrecursive, constrained and function-free. It is also a 2-literal definition, where the maximum arity of background knowledge predicates is 2 ($k = 2, j = 2$).

Training examples	*Background knowledge*		
$daughter(sue, ann).$ $\oplus$	$mother(ann, sue).$	$parent(X, Y) \leftarrow$	$female(ann).$
$daughter(eve, tom).$ $\oplus$	$mother(ann, tom).$	$mother(X, Y).$	$female(eve).$
$daughter(tom, ann).$ $\ominus$	$father(tom, eve).$	$parent(X, Y) \leftarrow$	$female(sue).$
$daughter(eve, ann).$ $\ominus$	$father(tom, jim).$	$father(X, Y).$	

Table 1. A simple ILP problem: learning family relationships.

Given the above definitions, we can state the following prior results. Page and Frisch [11] have shown that a single constrained, nonrecursive definite program clause is PAC-learnable. Džeroski and Lavrač [4] have shown that the problem of learning constrained nonrecursive function-free program clauses can be transformed into a propositional learning problem. Džeroski et al. [5] prove that k-clause determinate function-free nonrecursive definitions are PAC-learnable under simple distributions.

The work reported here is based on the latter results, and extends the former, replacing the single constrained nonrecursive definite clause with a set of nonrecursive constrained function-free program clauses. The class of constrained programs considered in this paper is a proper subset of the class of determinate programs considered in [5]. However, the learnability results for determinate programs are *distribution-dependent*, i.e., they hold for the class of simple distributions only, whereas the results presented here are *distribution-free.*

3 Transforming ILP problems to propositional form

Our learnability results are based on transforming an ILP problem to a propositional form, then using propositional learnability results. We consider only the problem of learning nonrecursive constrained function-free program clauses. To solve the problem of constructing a definition for the target predicate $q(X_1, X_2, ..., X_n)$ by transforming it to propositional form proceed as follows.

Algorithm 1

Input: *Examples for target predicate* $q(X_1, X_2, \ldots, X_n)$ *and definitions of predicates* $p_1, p_2, ..., p_l$.

Output: *A propositional learning task, i.e., a set of features and a set of examples (vectors of feature values).*

1. $F = \{\ p_r(Y_1, Y_2, ..., Y_{r_j}) | p_r \in \{p_1, ..., p_l\}, Y_i \in \{X_1, ..., X_n\}\}$
2. *for each* $q(a_1, a_2, ..., a_n) \in \mathcal{E}^+$ *and each* $\neg q(a_1, a_2, ..., a_n) \in \mathcal{E}^-$ *do*
   - *set the values of* $X_1, ..., X_n$ *to* $a_1, ..., a_n$,
   - *determine* f, *the vector of truth values of the literals in* F, *by posing the corresponding ground queries from* F *to the background knowledge,*
   - f *is an example of a propositional concept* c *(positive if* $q(a_1, a_2, ..., a_n)$ *or negative if* $\neg q(a_1, a_2, ..., a_n)$*)*

First, construct a list F of all literals that use predicates from the background knowledge and variables from the set $\{X_1, X_2, ..., X_n\}$. This is the list of features used for propositional learning. Second, transform the examples to propositional form. For each example, the truth value of each of the propositional features is determined by calls (queries) to the background knowledge. These two steps are done by Algorithm 1.

Next, apply a propositional learning algorithm to the propositional version of the problem. Finally, transform the induced propositional concept definition to program clause form. In this step, each feature in the propositional description is replaced with the corresponding literal.

The background knowledge $\mathcal{B}$ may take the form of a set of ground facts or a nonground logic program. Only ground (membership) queries have to be posed to $\mathcal{B}$. (NB: These are not queries to the example distribution!) The arguments of the target and background knowledge predicates may be sorted (or typed), as in LINUS [7], in which case the number of propositional features involved is greatly reduced [4].

To illustrate the transformation process, Table 2 gives the propositional version of the ILP problem from Table 1. In Table 2, d, f and p stand for *daughter*, *female* and *parent*, respectively. To outline the transformation from a propositional DNF formula to a predicate definition, suppose a propositional learner induces the concept $c \leftrightarrow x_1 \wedge x_5$ from the examples in Table 2. The feature x_1 is replaced with the literal $female(X)$, the feature x_5 with the literal $parent(Y, X)$ and the definition obtained is: $daughter(X, Y) \leftarrow female(X), parent(Y, X)$.

			Propositional features					
$d(X,Y)$	X	Y	$f(X)$	$f(Y)$	$p(X,X)$	$p(X,Y)$	$p(Y,X)$	$p(Y,Y)$
c			x_1	x_2	x_3	x_4	x_5	x_6
1	*sue*	*ann*	1	1	0	0	1	0
1	*eve*	*tom*	1	0	0	0	1	0
0	*tom*	*ann*	0	1	0	0	1	0
0	*eve*	*ann*	1	1	0	0	0	0

Table 2. Propositional form of the *daughter* relationship problem.

4 Results

Theorem 1. *Algorithm 1 transforms the ILP problem of learning a set of constrained nonrecursive function-free program clauses defined by a set of m examples $\mathcal{E}$ of the target predicate $q(X_1, X_2, ..., X_n)$, and background predicates $p_1, p_2, ..., p_l$ of maximum arity j, to a propositional form in $\mathcal{O}(poly(j)ml n^j)$, time, if each query to the background knowledge takes $\mathcal{O}(poly(j))$ to answer.*

Proof: The number of features in F (step 1 of the algorithm) is bounded by $N = ln^j$, as the arity of the background predicates is bounded by j, and for

each of the l background predicate there can be at most n^j features. The transformation of a single example to propositional form takes $\mathcal{O}(poly(j)N)$ time to determine the truth values of features in F. For m examples, the transformation process takes $\mathcal{O}(poly(j)mln^j)$ time. □

Theorem 2. *k-literal nonrecursive predicate definitions consisting of constrained function-free normal program clauses are polynomially PAC-learnable under arbitrary distributions.*

Proof: After transforming the problem to a propositional form, we use the algorithm for learning k-DNF outlined in [6]. The transformation from propositional k-DNF to a k-literal definition takes $\mathcal{O}(h)$ time, where h is the size of the induced propositional formula, which is at most $\mathcal{O}((2N)^{k+1})$ time. The learnability under arbitrary distributions of k-literal predicate definitions consisting of nonrecursive constrained function-free program clauses then follows from the polynomial PAC-learnability of k-DNF. □

5 Discussion and further work

We have proved a positive learnability result for a restricted class of logic programs. The complexity analysis indicates that our approach would scale well with the arity of the target predicate and the number of predicates in the background knowledge, but not with the maximum number of literals in a clause and the maximum arity of background knowledge predicates.

Despite the imposed restrictions, the considered class of logic programs includes many interesting and nontrivial concept definitions. These include concepts from chess, such as position illegality in a chess endgame [12, 7, 3], and qualitative models of dynamic systems [3]. Such concepts have been successfully induced by existing ILP systems.

Let us now discuss how these restrictions can be removed. Removal of the function-free restriction is straightforward, because any clause containing function symbols can be *flattened*, that is, rewritten in function-free form with the addition of one background clause per function symbol [13]. However, a constrained clause with function symbols can yield a flattened clause with new variables. Thus, the restrictions to constrained and function-free clauses are tightly coupled. Fortunately, the new variables introduced by the flattening process are determinate, i.e., have uniquely determined values, given the values of the old variables.

In a separate paper [5], we relax both restrictions and prove that k-clause function-free determinate predicate definitions are learnable under a broad class of distributions, called simple distributions [9]. However, the learnability results for the determinate (possibly recursive) case, require sampling according to the noncomputable universal distribution m [9], or its polynomial-time version, whereas the results presented here are distribution-free.

The restriction to non-recursive clauses is a more fundamental one. It can be relaxed if we allow the use of queries [5] about the target predicate. These

results are also derived for simple distributions under the assumption of sampling according to the universal distribution m. Further research should concentrate on removing the determinacy restriction and the restriction to simple distributions.

Only positive results can be obtained using our transformation approach. Namely, the hardness of a transformed propositional problem does not guarantee that the original first-order problem cannot be solved efficiently by some other means. Page and Frisch [11] show positive results for constrained clauses, also showing negative results for sorted theories. An advantage of our approach, however, is that new propositional PAC-learnability results, such as [1], can be transferred to the ILP framework.

Acknowledgements
The work reported in this paper is part of the ESPRIT BRA Project 6020 Inductive Logic Programming. Thanks to Luc De Raedt for his comments on the paper.

References

1. D. Angluin, M. Frazier and L. Pitt. Learning conjunctions of Horn clauses. *Machine Learning*, 9(2): 147–164, 1992.
2. L. De Raedt. *Interactive Theory Revision: An Inductive Logic Programming Approach.* Academic Press, London, 1992.
3. S. Džeroski and I. Bratko. Handling noise in inductive logic programming. In *Proc. Second International Workshop on Inductive Logic Programming.* ICOT TM-1182, Tokyo, 1992.
4. S. Džeroski and N. Lavrač. Refinement graphs for FOIL and LINUS. In S.H. Muggleton, editor, *Inductive Logic Programming*, pages 319–333, Academic Press, London, 1992.
5. S. Džeroski, S. Muggleton and S. Russell. PAC-learnability of determinate logic programs. In *Proc. Fifth ACM Workshop on Computational Learning Theory*, pages 128–135, ACM Press, Baltimore, MD, 1992.
6. D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's model. *Artificial Intelligence*, 36(2): 177-221, 1988.
7. N. Lavrač, S. Džeroski and M. Grobelnik. Learning nonrecursive definitions of relations with LINUS. In *Proc. Fifth European Working Session on Learning*, pages 265–281, Springer, Berlin, 1991.
8. J. W. Lloyd. *Foundations of Logic Programming* (2nd edn), Springer, Berlin, 1987.
9. M. Li and P. Vitányi. Learning simple concepts under simple distributions. *SIAM Journal of Computing*, 20(5): 911–935, 1991.
10. S. H. Muggleton. *Inductive Logic Programming*, Academic Press, London, 1992.
11. C. D. Page and A. M. Frisch. Generalization and learnability: a study of constrained atoms. In S. H. Muggleton, editor, *Inductive Logic Programming*, pages 29–61, Academic Press, London, 1992.
12. J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3): 239–266, 1990.
13. C. Rouveirol. Extensions of inversion of resolution applied to theory completion. In S.H. Muggleton, editor, *Inductive Logic Programming*, pages 63–92, Academic Press, London, 1992.
14. E. Y. Shapiro. *Algorithmic Program Debugging*, The MIT Press, Cambridge, MA, 1983.