Properties of Inductive Logic Programming in Function-Free Horn Logic

Irene Stahl

Fakultät Informatik, Universität Stuttgart, Breitwiesenstr. 20-22, D-70565 Stuttgart

Abstract. Inductive Logic Programming (ILP) deals with inductive inference in first order Horn logic. A commonly employed restriction on the hypothesis space in ILP is that to function-free programs. It yields a more tractable hypothesis space, and simplifies induction. This paper investigates basic properties of ILP in function-free languages.

1 Introduction

Because of the limitations of propositional learning algorithms there is an increasing interest in investigating learning methods in a first order framework. *Inductive Logic Programming (ILP)* [Mug92] is one of the approaches that received a lot of attention recently. The task of ILP is to inductively learn logic programs from examples in presence of background knowledge.

In order to constrain the generally infinite hypothesis space, ILP-systems impose restrictions, so-called *biases*, on their hypothesis language. These include for example the vocabulary or syntactic form of the target clauses. A commonly employed restriction in ILP is that to *function-free* Horn logic. Though more expressive than propositional logic, it still allows for deciding logical implication. These advantages make function-free languages prominent not only in ILP, but also in deductive databases and knowledge representation.

In this paper, we explore basic properties of ILP in function-free languages. First, we prove the decidability of the learning problem in function-free logic. Then, we investigate flattening as means to transform programs in function-free form, and discuss its limitations for inductive inference.

2 Basic Definitions

The task of ILP is defined formally as follows. Given ground facts $E^{\oplus}$ and $E^{\ominus}$ as positive and negative examples, a logic program B as background knowledge and a target language L with finitely many predicate symbols, find a logic program $P \in L$ such that $B \cup P \vdash E^{\oplus}$ (*completeness*) and $B \cup P \not\vdash E^{\ominus}$ (*consistency*). The quadruple $(E^{\oplus}, E^{\ominus}, B, L)$ is called the *learning problem*. Deciding whether a solution P exists is called the *FA- (finite axiomatisability) problem.*

If L is function-free, it contains no functions of arity ≥ 1. However, it may contain constants. As each program contains only finitely many constants, its Herbrand base is finite. This allows to decide whether a fact is implied by the program. Apart from the decidability which allows to check hypothesis on completeness and consistency, function-free logic simplifies the description and implementation of learning operators.

3 Decidability of the FA-problem in Function-Free Logic

The restriction to finitely many constants is fundamental for function-free languages. It leads to an interesting observation when inductive inference is concerned. If all n constants in B, $E^{\oplus}$ and $E^{\ominus}$ are known, it suffices to consider clauses with at most n variables for the target program. As these finitely many programs can be enumerated and tested on completeness and consistency, the FA-problem is decidable. The following theorem captures the above observation.

Theorem 1. *Given a function-free language L with n different constants, then for each P in L there exists a P' in L such that each clause in P' contains at most n variables, and $P \vdash a$ iff $P' \vdash a$ for each fact a in L.*

Proof. [Rei93] Each clause $C \in P$ with $m > n$ variables is replaced by n^m clauses $C\sigma$ for each possible substitution $\sigma : vars(C) \rightarrow \{Z_1, .., Z_n\}$. For the resulting program P' we show that $P \vdash a \Leftrightarrow P' \vdash a$.
$' \Rightarrow'$: Without loss of generality we assume $P \vdash a$ via a SLD proof

$$((..((\overline{a} \cdot C_1\theta_1) \cdot C_2\theta_2)..) \cdot C_k\theta_k)^1.$$

Then, $((..((\overline{a} \cdot C_1\theta_1..\theta_k) \cdot C_2\theta_2..\theta_k)..) \cdot C_k\theta_k)$ is also a proof that $P \vdash a$. Given a substitution ρ which substitutes all variables in $C_i\theta_i..\theta_k$ with an arbitrary constant, $((..((\overline{a} \cdot C_1\theta_1..\theta_k\rho) \cdot C_2\theta_2..\theta_k\rho)..) \cdot C_k\theta_k\rho)$ is also a proof. Now $C_i\theta_i..\theta_k\rho$ is a ground clause with at most n different constants so that there exists a $C_i' \in P'$ and a substitution ρ_i' such that $C_i\theta_i..\theta_k\rho = C_i'\rho_i'$. Thus, $((..((\overline{a} \cdot C_1'\rho_1') \cdot C_2'\rho_2')..) \cdot C_k'\rho_k')$ is a proof.
$' \Leftarrow'$: We assume $P' \vdash a$ via a SLD-proof

$$((..((\overline{a} \cdot C_1'\theta_1') \cdot C_2'\theta_2')..) \cdot C_k'\theta_k').$$

For each C_i' there is a $C_i \in P$, either $C_i = C_i'$ if C_i contains $\leq n$ variables, or $C_i\sigma = C_i'$. Therefore, C_i' can be replaced by C_i and θ_i' through $\sigma\theta_i'$ without changing the success of the proof.

The decidability of the FA-problem in function-free logic depends on whether all constants are known. This will in general not be the case, especially if cross-validation is used. This technique presents only a part of the examples as training set to the learning method. The number of new constants in the remaining test set is unknown, and likewise the upper bound for the number of variables in the target clauses.

However, theorem 1 can be generalised to the case that $E^{\oplus}$ and $E^{\ominus}$ contain constants not in B or L. The generalisation is based on the subsumption theorem [Rou91]. A program P implies a ground fact e with constants not in P *if and only if* P implies the fact e' that results from replacing these constants by variables. That is, the unknown constants themselves do not matter, but only their potential number within an example. This number is bound by the maximum predicate arity max_A in $E^{\oplus}$ and $E^{\ominus}$. Thus, if L is missing some constants in $E^{\oplus}$ and $E^{\ominus}$, it suffices to consider clauses with at most $n + max\_A$ variables for

[1] Here, $(A \cdot B\theta)$ is the result of resolving A and B with substitution θ.

the target program. That is, even in case that examples with unknown constants are to be covered, the FA-problem is decidable.

An interesting question is whether inducing programs that cover examples with new constants is really desirable. Due to the subsumption theorem, covering examples with new constants means that the according allquantified formula is implied. This is often too strong such that many systems require that knowledge about all constants in $E^{\oplus}$ and $E^{\ominus}$ is present in B, e.g. [Qui90]. Even more, for the case that new examples contain constants missing in B, techniques to acquire the background knowledge about them have been proposed [Rae91].

However, this technique leads to a stronger success criterion for learning, and accordingly to the undecidability of the FA-problem. The induced program must cover not only the given examples with respect to the background knowledge, but also new examples with respect to an *augmented* background knowledge.

4 Transformation to a Function-Free Form

In order to obtain the advantages of function-free logic without sacrificing the expressiveness of unrestricted Horn logic, a representation change called *flattening* has been proposed in [Rou91].

Flattening transforms programs to function-free form by replacing n-ary terms with predicates of arity $n+1$. Given a clause C, each occurrence of a term $f(t_1,...,t_n)$ is replaced by a variable X, and a new literal $f_p(t_1,...,t_n,X)$ is added to the body of C. The predicate f_p is defined by the unit clause $f_p(t_1,...,t_n,f(t_1,...,t_n))$. Flattening is a reversible process. Removing all predicates $f_p(t_1,...,t_n,X)$ from a flat clause, and unifying X with $f(t_1,...,t_n)$ yields the original clause. A program is equivalent to its flattened counterpart.

Theorem 2. [Rou91] *If $flat(P)$ is the flattened, function-free version of P, and $flat\_defs(P)$ the according definitions of the flattening predicates, then* $P \vdash A$ iff $flat(P) \cup flat\_defs(P) \vdash flat(A)$

However, this is not completely the desired result, as $flat\_defs(P)$ still contains structured terms. The really desirable result would be $P \vdash A$ iff $flat(P) \vdash flat(A)$ or, equivalently [Rou91] $P \vdash A$ iff $flat(P) \cup skolemized\_body(flat(A)) \vdash skolemized\_head(flat(A))$. And this is in fact the result that is used in the system ITOU [Rou91] for the subsumption test. However, the equivalence is not generally valid, as the following example will show.

Example 1. Let P be
$$succ(0,s(0))$$
$$succ(s(X),s(Y)) \leftarrow succ(X,Y)$$
$$p(X,Y) \leftarrow succ(s(s(X)),Z), succ(s(s(Y)),Z)$$
and let A be $p(0,0)$. The corresponding flat version of P is

$flat(P)$	$flat\_defs(P)$
$succ(N,SN) \leftarrow 0_p(N), s_p(N,SN)$	$0_p(0)$
$succ(SX,SY) \leftarrow s_p(X,SX), s_p(Y,SY), succ(X,Y)$	$s_p(X,s(X))$
$p(X,Y) \leftarrow s_p(SX,SSX), s_p(SY,SSY), s_p(X,SX),$ $s_p(Y,SY), succ(SSX,Z), succ(SSY,Z)$	

and $flat(A) = (p(N,N) \leftarrow 0_p(N))$.

Then, $P \vdash A$ holds. But contrary to the equivalence assumed in ITOU, $flat(P) \not\vdash flat(A)$, or equivalently $flat(P) \cup \{0_p(sk_0)\} \not\vdash p(sk_0, sk_0)$, as the predicate s_p has no positive occurrence in $flat(P)$.

That is, the equivalence tacitly assumed for the subsumption test in ITOU is generally not valid. Only if $flat\_defs(P)$ or, alternatively, arbitrarily many constants are supplied, the equivalence holds.

5 The Weakness of Function-Free Logic

The expressiveness of a function-free language is determined by its inventory of constants and predicates. Particularly the finite set of constants leads to a finite set of potential target programs. A weakness of function-free logic is that the range of expressible concepts is fixed even if additional predicates are introduced.

Introducing new predicates or *predicate invention* is performed to extend the vocabulary in case that the target language is insufficient for the learning task. New predicates generally increase the expressiveness of a language. However, because of the restricted expressiveness of function-free languages, new predicates cannot exceed the given predicates. The uselessness of predicate invention for recovering from a failure of the learning problem in function-free logic is proved in [Sta94]. It contrasts the power of predicate invention for enlarging the range of expressible concepts in the general case.

6 Conclusions

Function-free languages play a prominent role under the biases of ILP-systems. This paper investigates basic properties of ILP in function-free languages. The main results are the decidability of the learning problem and the according weakness of function-free logic as target language. This weakness turns out particularly in the uselessness of predicate invention as bias shift operation.

Acknowledgements This work has been supported by the ESPRIT BRA 6020 ILP. I want to thank Klaus Reinhardt for his ideas concering the proofs.

References

[Mug92] Muggleton, S. (1992): *Inductive Logic Programming*, in S. Muggleton (ed): Inductive Logic Programming, Academic Press

[Qui90] Quinlan, J. R. (1990): *Learning Logical Definitions from Relations*, Machine Learning 5

[Rae91] De Raedt, L., Feyaerts, J., Bruynooghe, M. (1991): *Acquiring Object-Knowledge for Learning Systems*, in Y. Kodratoff (ed): Proceedings of the Fifth European Working Session on Learning, Springer

[Rei93] Reinhardt, K. (1993), personal communication

[Rou91] Rouveirol, C. (1991): *ITOU: Induction of First Order Theories*, in S. Muggleton (ed): Inductive Logic Programming, Academic Press

[Sta94] Stahl, I. (1994): *On the Utility of Predicate Invention in Inductive Logic Programming*, this volume