An algorithm for inducing least generalization under relative implication

Svetla Boytcheva
Department of Information Technologies, Faculty of Mathematics and Informatics, Sofia University
5 J. Bauchier Blvd., 1164 Sofia, Bulgaria
svetla@fmi.uni-sofia.bg

Zdravko Markov
Department of Computer Science, Central Connecticut State University
1615 Stanley Street, New Britain, CT 06050, USA
markovz@ccsu.edu

Abstract

Inductive Logic Programming (ILP) deals with inducing clausal theories from examples basically through generalization or specialization. The specialization and generalization operators used are mainly based on three generality orderings - subsumption, implication and implication relative to background knowledge. Implication is stronger than subsumption, but relative implication is more powerful because background knowledge can be used to model all sorts of useful properties and relations. The least generalization under relative implication (LGRI) does not exist in the general case, but it exists if both the set to be generalized and the background knowledge satisfy some special conditions. The present paper discusses an algorithm for computing LGRI in cases when the latter exists.

Introduction

Inductive Logic Programming (ILP) investigates the problem of inducing clausal theories from given sets of positive and negative examples. An inductively inferred theory must imply all of the positive examples and none of the negative examples. The problem to find the least generalization of a set of clauses under implication relative to background knowledge is one of the fundamental problems related to the ILP task.

In the present paper we discuss the problem of finding generalizations of sets consisting of positive-only examples represented in the language of function-free Horn clauses with respect to background knowledge, represented in the same language. We also want the generalized hypothesis *H* to be represented in the same language.

If the set of clauses *S* to be generalized and the background knowledge Σ contain general clauses, the question of existence of LGRI has a negative answer. Even if *S* and Σ are both finite sets of function-free clauses, a LGRI of *S* relative to Σ does not necessarily exist. A counter example illustrating this is suggested in (Nienhuys-Cheng and de Wolf 1996, 1997). There are cases however, where by introducing some restrictions on both *S* and Σ we achieve the existence of LGRI (Boytcheva 2000). One of these restrictions is the so-called *utter model*, defined below.

Definition 1 (Utter model). Let $\Sigma = \{C_1,\ldots,C_k\}$ and $S = \{D_1,\ldots,D_n\}$ be sets of definite clauses. *S* has an *utter model w.r.t.* Σ, if for each clause D_i and each body literal in *L* in D_i, there exists a clause *E* either from *D* or from *S* and a substitution σ, such that $L\sigma \in E$.

The theoretical basis of our algorithm for inducing LGRI is the existence theorem, stated below and proven in (Boytcheva 2000).

Theorem 1. Let $\Sigma = \{C_1,\ldots,C_k\}$ be a finite set of function-free definite program clauses and $S = \{D_1,\ldots,D_n\}$ be a set of *function-free* definite program clauses, where all D_i have the *same predicate symbol* in their heads and at least one of them is *non-tautologous*. If *S* has an *utter model* under Σ and all clauses in *S* are *generative* (all head variables appear also in the body), then there exists *H* which is a LGRI of *S* relative to Σ, i.e. $H \models_{\Sigma} S$.

The present paper is organized as follows. The following section discusses some related work. Then we present some basic definitions used in the further discussion. The main result shown in the paper is our algorithm for computing least generalizations relative to implication (LGRI). We discuss some properties of this algorithm and illustrate it with several examples. Finally we conclude with a discussion of future research ideas.

Related work

The approach of least generalizations is usually considered within the more general framework of *relational learning* – an area of machine learning dealing with relational representation languages for the examples and hypotheses. The issues of relational learning have been discussed in some early papers by (Plotkin 1970; Vere 1980; Dietterich and Michalski 1981; Langley 1985). The basic problem stated in these and other papers is the design of *efficient methods for searching the hypothesis space* usually in a Horn clause representation language. The approaches to solving this problem can be summarized as follows:

- Using higher-order rule schemas to constrain search. This is the approach taken in MODELER (Wrobel 1988) and its extensions CIA (De Raedt and Bruynooghe, 1989a, 1989b) and RDT (Kietz and Wrobel, 1991).
- Using search heuristics based on information gain –an approach, taken in FOIL (Quinlan 1990) and its extensions CHAM (Kijsirikul, Numao and Shimura 1991), FOCL (Pazzani and Kibler 1990) and STRUCT (Watanabe and Rendell 1991).
- Constraining search by inducing refutation trees with inverse resolution operators. The approach is used in systems as MARVIN (Muggleton and Buntine 1988), RINCON (Wogulis and Langley 1989; Wogulis 1989), CIGOL (Muggleton and Buntine 1988), IRES (Rouveirol and Puget 1990) and SIERES (Wirth and O'Rorke 1991).
- Using propositional or other representations to simplify the search space. Basic instances of this approach are LINUS (Lavrač, Dzeroski and Grobelnik 1991) and Wy1 (Flann and Dietterich 1986).
- Avoiding general search by constructing relative least general generalizations. This is the approach taken in GOLEM (Muggleton and Feng 1990). Our approach falls in the same category.

As generally the relational languages are complex and consequently the search space is usually huge, some approaches reduce the relational language to propositional representations, while others try to impose constraints on the relational representation. The latter approaches are more promising, because they preserve the expressive power of the relational languages.

Various restrictions have been suggested to simplify the search in the relational hypothesis space. In many of the approaches the background knowledge consists only of ground literals clauses. Our approach however departs from these strong restrictions. The algorithm we introduce hereafter can work with function-free non-recursive Horn clauses without negation. The algorithm also exhibits a kind of a predicate invention feature - if necessary it generates new-predicate description to extend the background knowledge.

Preliminaries

In this section we give a theorem and a basic definition from (Lloyd 1984), both needed in the discussion of the algorithm in the next section.

Theorem 2 (Unification theorem). Let *S* be a finite set of first order expressions. If *S* is unifiable, then the unification algorithm terminates and gives an *mgu* for *S*. If *S* is not unifiable, then the unification algorithm terminates and reports this fact.

Definition 2 (Disagreement set). Let *S* be a finite set of first order expressions. The *disagreement set* of *S* is defined as follows. Locate the leftmost symbol position at which not all expressions in *S* have the same symbol and extract from each expression in *S* the sub-expression beginning at this symbol position. The set of all such sub-expressions is the disagreement set.

Algorithm for inducing LGRI

Let $S = \{D_1, \ldots, D_n\}$ be a finite set of function-free Horn clauses with identical head literals *Head*, i.e. D_i=*Head*:- $Body_i$. Let $\Sigma = \{C_1, \ldots, C_k\}$ be a finite set of function-free Horn clauses. Let *S* and Σ satisfy the requirements of Theorem 1. Then, there exists *H*, a LGRI of *S* relative to Σ, where the clauses in *H* have the head literal *Head*.

Algorithm

1. Using the unification algorithm find the common literals in the bodies of clauses from *S* and create a set *Common*.
2. Using the unification algorithm find the disagreement set of *S* and create set $S' = \{F_i \mid F_i = Body_i \backslash Common\ \theta_i, i=1,\ldots,n\}$, where *Common* is the set from step 1, $Body_i$ is the body of the clause $D_i \in S$ and θ_i is the most general unifier of *Common* and $Body_i$.
3. If *S'* is empty then go to **step 11** else continue.
4. Using Σ and *S'* create a set *R* containing all resolvents of clauses from *S'* and Σ. This set is finite, because all clauses are non-recursive and function-free, and *S* and Σ are finite sets.
5. If *R* is empty then go to **step 9** else continue.
6. Replace each $F_i \in S'$ (or a part of it) with its corresponding resolvent R_{ij} from *R*.
7. Create a set *R'***,** where:

   $R' = \{R_{ij} \mid R_{ij} \in R, \exists F_i \in S', \exists G_i \subseteq F_i, \exists \sigma_i, R_{ij} = G_i\sigma_i\}$
8. Use the unification algorithm to create a set *RCommon* containing the common literals in the clauses from *R'*.
9. Create a set *S''* = *S'* \ *RCommon* and separate the literals in the clauses of *S''* in groups with independent variables. Identify a subset of literals E_i in each group, such that, its variables when substituted by the most general unifier, map onto a common set of variables *V* for all clauses in *S''*. Find all such sets (if exist).
10. For each non-empty set *V* use its variables as head variables and generate a new clause *New(V)* :- E_i with new predicate symbol *New* and go to **step 12**.
11. Return *H* = *Head* :- *Common*. **END**
12. Return $H = \{H_1, \ldots, H_m\}$, where

    H_i = *Head* :- *Common*, *RCommon*, New_i.

    (New_i (i = 1,…,m) are the heads of the predicates generated in step 10.) **END**

Annotated example

For better understanding we illustrate the steps of the algorithm with an example. Consider the following two DCG rules, represented as Prolog clauses:

S = {sentence(X) :- append(L1,L2,X), append(D,N,L1),
det(D), noun(N), verb(L2);
sentence(X) :- append(L1,L2,X), proper_name(L1),
verb(L2) }

$\Sigma = \{\}$

Before starting the algorithm we need to rename the variables in the two clauses. This is a standard procedure required for the application of the resolution rule. Thus we get:

S = {
sentence(X1) :- append(L11,L21,X1),append(D1,N1,L11),
det(D1), noun(N1), verb(L21);
sentence(X2) :- append(L12,L22,X2), proper_name(L12),
verb(L22) }

Step 1. First, we antiunify the two heads in S and determine the head literal of the hypothesis *sentence(A)* along with the substitutions {A/X1} and {A/X2}. Then we find the set *Common* = {append(B,C,A), verb(C)} and the substitutions θ_1 = {A/X1, B/L11, C/L21} and θ_2 = {A/X2, B/L12, C/L22}.

Step 2. We compute the disagreement set S' = {F_1, F_2}, where F_1 = append(D1,N1,B), det(D1), noun(N1) and F_2 = proper_name(B).

Step 3. S' is not empty, so we try step 4.

Steps 4, 5. Since Σ is empty we cannot create any resolvents. Thus R is also empty and we continue with step 9.

Step 9. Since *RCommon* is empty, $S'' = S'$. Both F_1 and F_2 have one group of dependent variables (no groups with mutually independent variables). These groups are {D1, N1,B} and {B}. Obviously, the set of common variables V is {B}. The corresponding sets of literals are E_1 = {append(D1,N1,B), det(D1), noun(N1)} and E_2 = {proper_name(B)}.

Step 10. Here we create the new predicate

new(B) :- append(D1,N1,B), det(D1), noun(N1).
new(B) :- proper_name(B).

and continue with step 12.

Step 12. Return the LGRI of S relative to Σ:
H = sentence(A) :- append(B,C,A), verb(C), new(B).

Obviously the meaning of the new predicate is the definition of the noun phrase rule, so we can rename it accordingly as:

noun_phrase(B) :- append(D1,N1,B), det(D1), noun(N1).
noun_phrase (B) :- proper_name(B).

Correctness

Hereafter we shall prove the *correctness* of the algorithm. Firstly, we have to show that H is a *generalization* of S relative to Σ, i.e. $H \models_\Sigma S$. In fact, we have to prove that $H \cup \Sigma \models S$ or $H \cup \Sigma \models D_i$ for every D_i. Let H = {H_1,…,H_m}, where H_i is constructed by the algorithm as follows: H_i = *Head* :- *Common*, *RCommon*, *New*$_i$. Also, according to the construction method of *Common* and F_i, it is clear that *Body*$_i$ = *Common*, F_i. That is D_i = *Head* :- *Common*, F_i. For each F_i we have its resolvent with Σ in *RCommon*, which in turn is a part of H. In this way we have $H \cup \Sigma \cup \neg\{D_i\} \models \square$, that is $H \cup \Sigma \models D_i$.

Showing that H is a *least* generalization of S relative to Σ is needs more space and due to the lack of the latter we omit this proof here.

Applications

The algorithm can be used as an incremental algorithm for supervised learning where the teacher chooses a particular sequence of clauses from S and feeds them one at a time into the algorithm to be generalized (Figure 1). Another scenario is shown in Figure 2, where the algorithm generalizes a set of clauses directly.

The algorithm is also applicable to a situation without background knowledge, i.e. $\Sigma = \emptyset$. In this case it finds the least general generalization under implication of a set of clauses.

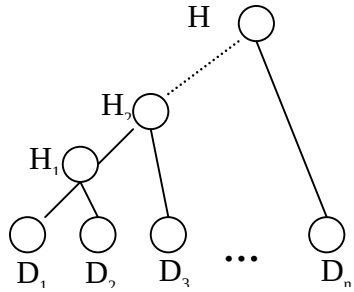

Fig. 1: Incremental construction of hypothesis.

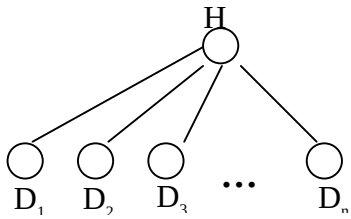

Fig. 2: Construction of a hypothesis from a set of clauses.

More examples

In this section we further illustrate the algorithm introduced in the previous section with more examples.

Example 1

Given

S = {grandparent(X,Z) :- man(X),parent(X,Y),parent(Y,Z);
grandparent(X,Z) :- parent(X,Y),man(Y),parent(Y,Z)}
Σ ={}

the algorithm generates the hypothesis

H = grandparent(X,Z) :- parent(X,Y), parent(Y,Z).

The set *Common* here is {parent(X,Y), parent(Y,Z)}. As no common set of variables can be found in the disagreement set S'= {F_1 = man(X), F_2 = man(Y)}, no new predicates are generated and the final hypothesis contains the set *Common* only.

The generalization in this example is equivalent to the one produced under the subsumption generality ordering, because H is a subset of each one of the two clauses in S.

Example 2

In this example we illustrate how the algorithm handles background knowledge. Given

S = {cuddlypet(X) :- small(X), fluffy(X), dog(X);
cuddlypet(X) :- fluffy(X), cat(X)},
Σ = {pet(X):-cat(X); pet(X):-dog(X); small(X):-cat(X)},

the algorithm infers

H = cuddlypet(X) :- fluffy(X), small(X), pet(X).

During this inference three other sets of clauses are generated:
Common = {fluffy(X)}
S' = {:-small(X),dog(X); :- cat(X)}
R = *RCommon* = {:-small(X), :-pet(X)}

Example 3

This example illustrates the most general situation. Given a set of clauses S and background knowledge Σ:

S = {employer(X) :- english(X), educated(X,Y),
good_reputation(Y);
employer(X) :- french(X), recommendation(X,Y),
previous_employer(X,Y)}

Σ = {nationality(X) :- english(X);
nationality(X) :- french(X)}.

the algorithm infers

H = {employer(X) :- nationality(X), good_skills(X,Y) }.

It also generates two clauses with new predicate symbols, which we replace for clarity with "good_skills":

good_skills(X,Y) :- educated(X,Y), good_reputation(Y).
good_skills(X,Y) :- recommendation(X,Y),
previous_employer(X,Y).

Conclusion and further work

The presented algorithm is a step towards investigating the potential of constructive generalization with respect to background knowledge in the context of Inductive Logic Programming. The problem of using logical implication in a constructive way is known to be hard. This especially applies to the computational complexity of the generalization operators. Our algorithm is based on two other algorithms, which are very well studied with respect to their complexity. These are the unification algorithm and the resolution procedure. In our experimental implementation of the algorithm we use *Prolog*, a language in which these algorithms are built-in. Thus the examples we discussed in the paper, as well as more complex examples run relatively quickly. Nevertheless we have to further investigate the computational complexity of our algorithm, because it uses unification and resolution in a specific way and under some restrictions that may allow to improve the algorithm overall efficiency.

Another direction for future research is extending the representation language to clauses with negation and/or recursive clauses. Also we will continue our theoretical investigations on other possible cases of existence of LGRI, especially for sets of general (non function-free) clauses.

References

Boytcheva, S. 2000. Least Generalization under Relative Implication. In *Proceedings of the Ninth International Conference Artificial Intelligence: Methodology, Systems and Applications*, 59-68. LNAI 1904: Springer-Verlag

De Raedt, L., & Bruynooghe, M. 1989a. Constructive induction by analogy. InProceedings of the Sixth International Workshop on Machine Learning , 476-477. Ithaca, NY: Morgan Kaufmann.

De Raedt, L., & Bruynooghe, M. 1989b. Constructive induction by analogy: A method to learn how to learn? In Proceedings of the Fourth European Working Session on Learning, 189-200. Montpellier, France: Pitman.

Flann, N. S., & Dietterich, T. G. 1986. Selecting appropriate representations for learning from examples. In *Proceedings of the Fifth National Conference on Artificial*

Intelligence, 460-466. Philadelphia, PA: Morgan Kaufmann.

Kietz, J-U., & Wrobel, S. 1991. Controlling the complexity of learning in logic through syntactic and task-oriented models. In Proceedings of the First International Workshop on Inductive Logic Programming , 107-126. Vienna de Castelo, Portugal: Unpublished.

Kijsirikul, B., Numao, M., & Shimura, M. 1991. Efficient learning of logic programs with non-determinate, non-discriminating literals. In Proceedings of the First International Workshop on Inductive Logic Programming 33-40. Vienna de Castelo, Portugal: Unpublished.

Van der Laag, P. 1995. An analyses of refinement operators in inductive logic programming, Ph.D thesis, Tinberg Institute Research Series, Rotherdam.

Lloyd, J.W. 1984. *Foundations of Logic Programming*, Springer-Verlag Berlin Heidelberg.

Muggleton, S. and Buntine, W. 1988. Machine invention of first-order predicates by inverting resolution. In J.Laird editior, *Proceedings of the 5th International Conference on Machine learning (ICML-88)*, pages 339-352, Morgan Kaufman, San Mateo, CA.

Muggleton, S., & Feng, C. 1990. Efficient induction of logic programs. *Proceedings of the First International Workshop on Algorithmic Learning Theory*, 368-381, Tokyo, Japan: Japanese Society for Artificial Intelligence.
Nienhuys-Cheng, S-H. and de Wolf, R. 1997. *Foundations of Inductive Logic Programming*, Springer-Verlag,Berlin Heidelberg.

Nienhuys-Cheng, S-H. and de Wolf, R. 1996. Least generalizations and greatest specializations of sets of clauses, *Journal of Artificial Intelligence*, 4:341-363.

Dietterich, T. G., & Michalski, R. S. 1981. Inductive learning of structural descriptions. *Artificial Intelligence*, 16, 257-294.

Langley, P. 1985. Learning to search: From weak methods to domain-specific heuristics. *Cognitive Science*, 9, 217-260.

Lavrač, N., Dzeroski, S., & Grobelnik, M. 1991. Learning non-recursive definitions of relations with LINUS . In *Proceedings of the Fifth European Working Session on Learning*, 265-281. Porto, Portugal: Springer-Verlag.

Pazzani, M., & Kibler, D. 1990. The utility of knowledge in inductive learning (Technical Report 90-18). University of California, Irvine, Department of Information and Computer Science.

Plotkin, G. D. 1970. A note on inductive generalization. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence* (Vol. V). New York, NY: Elsevier North-Holland.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning*, 5, 239-266.

Rouveirol, C., & Puget, J. F. 1990. Beyond inversion of resolution. In Proceedings of the Seventh International Conference on Machine Learning, 122-130. Austin, TX: Morgan Kaufmann.

Vere, S. A. 1980. Multilevel counterfactuals for generalizations of relational concepts and productions. *Artificial Intelligence*, 14, 139-164.

Wirth, R., & O'Rorke, P. 1991. Inductive completion of SLD proofs. *In Proceedings of the First In-Iternational Workshop on Inductive Logic Programming*, 167-176. Vienna de Castelo, Portugal:

Wogulis, J. 1989. A framework for improving efficiency and accuracy. In Proceedings of the Sixth Inter-national Workshop on Machine Learning (pp. 78{80). Ithaca, NY: Morgan Kaufmann.

Wogulis, J., & Langley, P. 1989. Improving efficiency by learning intermediate concepts. *In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 657-662, Detroit, MI: Morgan Kaufmann.

Wrobel, S. 1988. Automatic representation adjustment in an observational discovery system. In *Proceedings of the Third European Working Session on Learning*, 253-262, Glasgow, Scotland:Pitman.

Watanabe, L., & Rendell, L. 1991. Learning structural decision trees from examples. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence. Sydney, Australia: Morgan Kaufmann.