Integrating Abduction and Induction

Fabrizio Riguzzi[1]

Abstract. We propose an approach for the integration of abduction and induction in the context of Logic Programming. The integration is obtained by extending an Inductive Logic Programming system with abductive reasoning capabilities. In the resulting system, abduction is used to make assumptions in order to cover positive examples and avoid the coverage of negative ones. The assumptions generated can then be generalized in their turn.

The system provides a framework for solving tasks that are difficult or impossible for most Inductive Logic Programming systems, such as: learning from incomplete knowledge, learning abductive theories, learning exceptions, learning multiple predicates and learning normal logic programs.

1 Introduction

Lately, the relationship and integration between abduction and induction have received much attention and have been the subject of a series of ECAI and IJCAI workshops since 1996 [2]. In the context of Logic Programming, the two inference processes are studied, respectively, in the research fields of Abductive Logic Programming (ALP for short) [3] and Inductive Logic Programming (ILP for short) [7]. Even if the definitions of abduction and induction are still subject of debate, we will adopt the rather established definitions for abduction and induction that are given in ALP and ILP.

Abduction is the process of inferring explanations for observations. In ALP, explanations are obtained from *abductive theories* of the form $T = \langle P, A, IC \rangle$ where P is a logic program, A is a set of *abducible predicates* and IC is a set of *integrity constraints* in the form of denials. The problem of abduction can then be stated as follows [3]: given an abductive theory $T = \langle P, A, IC \rangle$ and an observation O, find an explanation Δ for the observation such that it contains only atoms of abducible predicates, it entails O together with P ($P \cup \Delta \models O$) and it is consistent with integrity constraints ($P \cup \Delta \models IC$). In this case we say that T *abductively entails* O with abductive explanation Δ and we write $T \models_A O$. In ALP, deafult negation is modelled through abduction by considering negative literals as new abducible literals of the form not_p and by including constraints of the form $\leftarrow p, not\_p$.

Induction, instead, is the process of inferring general rules from examples. In ILP, the problem of induction can be stated as follows [7]: given a logic program B (background knowledge), a set of positive and negative examples E^+ and E^- and a hypothesis space $\mathcal{P}$, find a logic program $P \in \mathcal{P}$ such that $B \cup P$ entails every positive example and no negative one.

We propose an approach for the integration of abduction and induction that consists in extending an Inductive Logic Programming system with abductive reasoning capabilities. The integration increases the power of both inference processes: abduction helps induction by generating atomic hypotheses that can be used as new training examples or to complete an incomplete background knowledge, while induction helps abduction by generalizing abductive explanations.

The algorithm we propose is presented in section 2 while the tasks that can be accomplished by it are discussed in section 3.

2 The Algorithm

The algorithm we present is an evolution of those proposed in [5, 6]. The algorithm solves a new learning problem where both background and target theory are abductive theories and abductive entailment is used in place of deductive entailment as the coverage relation.

Abductive Learning Problem: **given** an abductive theory $T = \langle P, A, IC \rangle$ as background theory, a set of positive and negative examples E^+ and E^- and a set $\mathcal{P}$ of possible programs, **find** a new abductive theory $T' = \langle P \cup P', A, IC \rangle$ such that $P' \in \mathcal{P}$ and $T' \models_A E^+, not\_E^-$, where $not\_E^- = \{not\_e^- | e^- \in E^-\}$ and E^+, not_E^- stands for the conjunction of each atom in E^+ and not_E^-.

The algorithm is obtained by extending a basic top-down covering ILP algorithm [7]. Abduction is used in order to cover examples: if a positive example can not be covered with the available information, assumptions are made in order to cover it. Moreover, assumptions of absence of atoms are made in order to ensure that negative examples are not covered. To this purpose, the coverage test of examples is performed by using the abductive proof procedure defined in [4] instead of the Prolog proof procedure. Each example is tested singularly but care is taken to ensure that explanations for different examples are consistent with each other by keeping a global set of assumptions.

Some of the target predicates can also be considered as abducible. In this case, after the generation of each clause, assumptions about target predicates are added to the training set, so that they become new training examples.

In the specialization loop, the space of clauses is searched by performing a beam search. The heuristic function has been specially designed in order to take into account abduction: an estimated accuracy has been used where example covered by making assumptions are given a lower weight with respect to

[1] DEIS, Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy friguzzi@deis.unibo.it

Submitted to ECAI 98
March 6, 1998

those covered without assumptions.

Note that we have considered a learning problem where no constraints are learned. In order to learn a full abductive theory, we learn first the rules, using the above algorithm, and then the constraints, using the system ICL [1] that learns from interpretations. The input interpretations for ICL are obtained from the set of assumptions generated in the rule learning phase.

3 Tasks

The system can be successfully applied to solve a number of tasks that are difficult or impossible for most ILP systems: learning from incomplete knowledge, learning abductive theories, learning exceptions, learning multiple predicates and learning logic programs with negation.

Let us show now an example of learning from incomplete knowledge. Consider the abductive background theory $B = \langle P, A, IC \rangle$ and training set:

$$
\begin{aligned}
P = \; & \{parent(john, mary), male(john), \\
& parent(david, steve), \\
& parent(kathy, ellen), female(kathy)\} \\
A = \; & \{male/1, female/1, not\_male/1, not\_female/1\} \\
IC = \; & \{\leftarrow male(X), not\_male(X) \\
& \leftarrow female(X), not\_female(X)\} \\
E^+ = \; & \{father(john, mary), father(david, steve)\} \\
E^- = \; & \{father(john, steve), father(kathy, ellen)\}
\end{aligned}
$$

From these data, the system learns the rule

$father(X, Y) \leftarrow parent(X, Y), male(X).$

that covers the positive examples and the negation of the negative ones by making the assumptions

$\Delta = \{male(david), not\_male(kathy)\}.$

At this point, we can either stop or go on and learn the constraints. The input to ICL consists of the positive interpretation $\{male(david)\}$ and of the negative one $\{male(kathy)\}$. From this input, ICL learns the constraint

$\leftarrow male(X), female(X).$

In order to learn exceptions to rules, when no literal can be added to a rule for a target predicate p/n in order to make it consistent, the rule is specialized by adding a new abducible literal $not\_abnorm_i(\vec{X})$. This addition transforms the rule into a default one that can be applied in all "normal" (or non-abnormal) cases. The refined rule becomes consistent by abducing $abnorm_i(\vec{t^-})$ for every negative example $p(\vec{t^-})$ previously covered. Positive examples, instead, will be covered by abducing $not\_abnorm_i(\vec{t^+})$ for every positive example $p(\vec{t^+})$ previously covered. These assumptions are then added to the training set, and are used to learn a definition for $abnorm_i/n$ that describes the class of exceptions. If there are exceptions to exceptions, the system adds a new literal not_abnorm_j/n to the body of the rule for $abnorm_i/n$ and the process is iterated. In this way, we are able to learn hierarchies of exceptions. Induction is used here to generalize the explanations generated by abduction.

The system can be used in order to learn multiple predicates and normal logic programs with minor modifications. When learning multiple predicates, covering top-down algorithms face the problem that adding a consistent clause to the theory can make previous clauses inconsistent. Therefore, each time a clause is added to the current hypothesis, the consistency of the whole hypothesis must be tested and retraction of previous clauses may be necessary. By employing abduction, we can avoid the cost of testing each time the consistency of the whole theory. In fact, by considering the negation of all target predicates as abducibles, when a clause is tested for consistency, all the negative literals for other predicates that are necessary to ensure the consistency are recorded by the abductive proof procedure in the set of assumptions. They then become new negative examples so that clauses learned afterwards will not make the previous clause inconsistent. Therefore, the global set of assumptions is used to store the information necessary to make the learning process monotonic with respect to consistency. The system must be extended so that it is able to recover from inconsistency when no clause can be found that is consistent with both the original and the generated negative examples. In this case, the system adds a clause that is consistent with the original negative examples only and retracts the previous clauses that generated the covered negative examples.

With similar reasoning, we can deal with the problem of learning normal logic programs. In this case, the problem of the covering algorithm is that adding a clause to the current hypothesis can cause the uncoverage of previously covered positive examples. Also here negated target predicates are considered as abducibles. Consider a clause that contains a negative literal in the body for the target predicate p/n. For each positive example covered by the clause, a negative example for p/n is generated. Afterwards, a rule for p/n will be learned that will be consistent with the generated negative examples thus not reducing the coverage of the first clause. In this case the set of assumptions is used to make the process monotonic with respect to positive example coverage and, as in the previous case, backtracking must be performed when the system is not able to find a consistent clause.

The system has been implemented in Prolog and is currently being used on a number of experiments on incomplete data from the marketing domain and on problems of learning multiple predicates and logic programs with negation.

REFERENCES

[1] L. De Raedt and W. Van Lear, 'Inductive constraint logic', in *Proceedings of the 5th International Workshop on Algorithmic Learning Theory*, (1995).

[2] *Abductive and Inductive Reasoning*, eds., Peter Flach and Antonis Kakas, Pure and Applied Logic, Kluwer, 1998.

[3] A.C. Kakas, R.A. Kowalski, and F. Toni, 'The role of abduction in logic programming', in *Handbook of Logic in AI and Logic Programming*, eds., D. Gabbay, C. Hogger, and J. Robinson, volume 5, 233–306, Oxford University Press, (1997).

[4] A.C. Kakas and P. Mancarella, 'On the relation between truth maintenance and abduction', in *Proceedings of the 2nd Pacific Rim International Conference on Artificial Intelligence*, (1990).

[5] A.C. Kakas and F. Riguzzi, 'Learning with abduction', in *Proceedings of the 7th International Workshop on Inductive Logic Programming*, (1997).

[6] E. Lamma, P. Mello, M. Milano, and F. Riguzzi, 'Integrating induction and abduction in logic programming', *Information Sciences*. To appear.

[7] N. Lavrač and S. Džeroski, *Inductive Logic Programming: Techniques and Applications*, Ellis Horwood, 1994.