A System for Learning Abductive Logic Programs

Evelina Lamma, Michela Milano, Fabrizio Riguzzi
DEIS, Università di Bologna, Viale Risorgimento 2
I-40136 Bologna, Italy,
Tel. +39 51 6443033, Fax. +39 51 6443073
{elamma,mmilano,friguzzi}@deis.unibo.it

Paola Mello
Dip. di Ingegneria, Università di Ferrara,
Via Saragat 1, 41100 Ferrara, Italy
pmello@ing.unife.it

Abstract

We present the system LAP for learning abductive logic programs from examples and from a background abductive theory. A new type of induction problem has been defined as an extension of the Inductive Logic Programming framework. In the new problem definition, both the background and the target theories are abductive logic programs and the coverage of examples is replaced by abductive coverage.

LAP is based on a top-down learning algorithm that has been suitably extended in order to solve the new induction problem. In particular, the testing of example coverage is performed by using the abductive proof procedure defined by Kakas and Mancarella [9]. Assumptions can be made in order to cover positive examples and rule out negative ones and these assumptions can be used as new training data. LAP can be applied for learning in presence of incomplete knowledge and for learning exceptions to classification rules.

Keywords: Abduction, Learning.

1 Introduction

Inductive Logic Programming (ILP) [3] is a research area covering the intersection of Machine Learning and Logic Programming. Its aim is to devise systems that are able to learn logic programs from examples and from a background knowledge. Recently, in this research area, a number of works have become to appear on the problem of learning non-monotonic logic programs [2, 6, 4, 13].

A particular attention has been given to the problem of learning abductive logic programs [8, 11, 12, 10] and, more generally, to the relation existing between abduction and induction and how they can integrate and complement each other [5, 7, 1]. We consider an approach to the integration of the two in which abduction helps induction by allowing to make assumption about unknown facts. In [8, 11] we defined a new learning problem similar to the Abductive Concept Learning framework [7]. In this new framework we generate an abductive logic program from an abductive background knowledge and from a set of positive and negative examples of the concepts to be learned. The resulting theory must abductively entail all the positive examples and the default negation of the negative ones.

Learning abductive logic programs is useful for a number of reasons. We can learn theories for domains where abduction is an effective problem solving strategy, e.g. diagnostic problems. We can learn exceptions to classification rules exploiting negation by default. Moreover, we can learn in domains in which there is incompleteness of the available information, either in the background knowledge or in the training set. This is a very frequent case in practice, because very often the data available is incomplete and/or noisy.

We present the system LAP that solves the new learning problem. The system is based on the theoretical work developed in [8, 11]. It is an extension of a basic top-down algorithm adopted in ILP [3]. The extended algorithm takes into account abducibles and integrity constraints, and relies on the proof procedure defined in [9] for abductive logic programs. The key idea is that the abductive proof procedure is used for the coverage process of positive and negative examples in substitution of the deductive proof procedure of logic programming. Moreover, the abduced literals can be used as new training data for learning definitions for the abducible predicates.

The paper is organized as follows: in section 2 we recall the main concepts of Abductive Logic Programming (ALP), ILP, and the definition of the abductive learning framework. In section 3 we present the algorithm for learning abductive rules. In section 4 we describe the application of the system to the problem of learning exceptions to rules. In section 5 we conclude and present the directions for future works.

2 Abductive and Inductive Logic Programming

2.1 Abductive Logic Programming

An *abductive logic program* is a triple $\langle P, A, IC \rangle$ where:

- P is a normal logic program;
- A is a set of *abducible predicates*
- IC is a set of integrity constraints in the form of denials, i.e.:

  $\leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_{m+n}$.

Negation as Failure is replaced, in ALP, by Negation by Default and is obtained in this way: for each predicate symbol p, a new predicate symbol not_p is added to the set A and the integrity constraint $\leftarrow p(\vec{X}), not\_p(\vec{X})$ is added to IC, where $\vec{X}$ is a tuple of variables.

Given an abductive program $AT = \langle P, A, IC \rangle$ and a formula G, the goal of abduction is to find a (possibly minimal) set of ground atoms Δ (*abductive explanation*) of predicates in A which together with P entails G, i.e. $P \cup \Delta \models G$. It is also required that the program $P \cup \Delta$ is consistent with respect to IC, i.e. $P \cup \Delta \models IC$. When there exist an abductive explanation for G in AT, we say that AT *abductively entails* G and we write $AT \models_A G$.

In [9] a proof procedure for abductive logic programs has been defined. This procedure starts from a goal and results in a set of consistent hypothesis (abduced literals) that together with the program allow to derive the goal. We have extended this proof procedure in order to allow for abducible predicates to have a partial definition. Some rules may be available for them and we can make assumptions about missing facts.

2.2 Inductive Logic Programming

The ILP problem can be defined as [3]:

Given:

a set $\mathcal{P}$ of possible programs
a set E^+ of positive examples
a set E^- of negative examples
a consistent logic program B such that
$B \not\vdash e^+$ for at least one $e^+ \in E^+$.

Find:

a logic program $P \in \mathcal{P}$ such that
$\forall e^+ \in E^+$, $B \cup P \models e^+$ (*completeness*)
$\forall e^- \in E^-$, $B \cup P \not\models e^-$ (*consistency*).

Let us introduce some terminology. The sets E^+ and E^- are called *training sets*. The program P that we want to learn is the *target program* and the predicates which are defined in it are *target predicates*. The program B is called *background knowledge* and contains the definitions of the predicates that are already known. We say that the program P *covers* an example e if $P \cup B \models e$. Therefore the conditions that the program P must satisfy in order to be a solution to the ILP problem can be expressed as "P must cover all positive examples and must not cover any negative examples". The set $\mathcal{P}$ is called the *hypothesis space*. The importance of this set lies in the fact that it defines the search space of the ILP system. In order to be able to effectively learn a program, this space must be restricted as much as possible. If the space is too big, the search could result infeasible.

The *language bias* (or simply *bias* in this paper) is a description of the hypothesis space. Many formalisms have been introduced in order to describe this space [3], we will consider only a very simple bias in the form of a set of literals which are allowed in the body of the clauses for the target predicates.

2.3 The New Learning Framework

We consider a new definition of the learning problem similar to Abductive Concept Learning (ACL) [7]. In this extended learning problem both the background and target theory are abductive theories and the notion of deductive coverage is replaced by abductive coverage.

Given

a set $\mathcal{P}$ of possible abductive programs
a set of positive examples E^+,
a set of negative examples E^-,
an abductive theory $AT = \langle T, A, IC \rangle$ as background theory.

Find

A new abductive theory $AT' = \langle T', A, IC \rangle \in \mathcal{P}$ where $T' \supset T$ such that $AT' \models_A E^+, not\ E^-$, where $not\_E^- = \{not\_e^- | e^- \in E^-\}$.

Given an example e, we say that AT' *abductively entails* e when $AT' \models_A e$. By requiring that the conjunction of the positive and the negation of negative examples is abductively entailed by the final theory, we ensure that the abductive explanations for different examples are consistent with each other.

The abductive program that is learned can contain new rules (eventually containing abducibles in the body) but not new abducible predicates and new integrity constraints.

3 An algorithm for Learning Abductive Logic Programs

We present the system LAP that is able to learn abductive logic programs. The algorithm is obtained from the basic top-down ILP algorithm [3], by substituting the usual notion of coverage of examples with the notion of *abductive coverage*.

The basic top-down inductive algorithm [3] learns programs by generating clauses one after the other and generates clauses by means of specialization. As the basic inductive algorithm, LAP is constituted by two nested loops: the covering loop (figure 1) and the specialization loop (figure 2). At each iteration of the covering loop, a new clause is generated such that it covers at least one positive example and no negative ones. The positive examples covered by the rule are removed from the training set and a new iteration of the covering loop is started. The algorithm ends when the positive training set becomes empty. The new clause is generated in the specialization loop: we start with a clause with an empty body, and we add a literal to the body until the clause does not cover any negative examples while still covering at

procedure LAP(
  inputs : E^+, E^- : training sets,
    $AT = \langle T, A, IC \rangle$: background abductive theory,
  outputs : H : learned theory, Δ : abduced literals)

$H := \emptyset$
$\Delta := \emptyset$
while $E^+ \neq \emptyset$ **do** (covering loop)
  GenerateRule(input: AT, H, E^+, E^-, Δ; output: $Rule, E^+_{Rule}, \Delta_{Rule}$)
  Move to E^+ all the positive literals of target predicates in Δ_{Rule}
  Move to E^- all the atoms corresponding to
    negative literals of target predicates in Δ_{Rule}
  $E^+ := E^+ - E^+_{Rule}$
  $H := H \cup \{Rule\}$
  $\Delta := \Delta \cup \Delta_{Rule}$
endwhile
output H

Figure 1: The covering loop

procedure GenerateRule(
  inputs : AT, E^+, E^-, H, Δ
  outputs : $Rule$: rule,
    E^+_{Rule} : positive examples covered by $Rule$,
    Δ_{Rule} : abduced literals

Select a predicate to be learned p
Let $Rule = p(X) \leftarrow true$.
repeat (specialization loop)
  select a literal L from the language bias
  add L to the body of $Rule$
  TestCoverage(in: $Rule, AT, H, E^+, E^-, \Delta$,
    out: $E^+_{Rule}, E^-_{Rule}, \Delta_{Rule}$)
  if $E^+_{Rule} = \emptyset$
    backtrack to a different choice for L
until $E^-_{Rule} = \emptyset$
output $Rule, E^+_{Rule}, \Delta_{Rule}$

Figure 2: The specialization loop

procedure TestCoverage(

 inputs : $Rule, AT, H, E^+, E^-, \Delta$

 outputs: E^+_{Rule}, E^-_{Rule}: examples covered by $Rule$

  Δ' : new set of abduced literals

$E^+_{Rule} = E^-_{Rule} = \emptyset$

$\Delta_{in} = \Delta$

for each $e^+ \in E^+$ **do**

 if AbductiveDerivation($e^+, \langle T \cup H \cup \{Rule\}, A, IC\rangle, \Delta_{in}, \Delta_{out}$)

   succeeds **then** Add e^+ to E^+_{Rule}; $\Delta_{in} = \Delta_{out}$

endfor

for each $e^- \in E^-$ **do**

 if AbductiveDerivation($not\_e^-, \langle T \cup H \cup \{Rule\}, A, IC\rangle, \Delta_{in}, \Delta_{out}$)

   succeeds **then** $\Delta_{in} = \Delta_{out}$

 else Add e^- to E^-_{Rule}

endfor

$\Delta_{Rule} = \Delta_{out} - \Delta$

output $E^+_{Rule}, E^-_{Rule}, \Delta_{Rule}$

Figure 3: Coverage testing

least one positive. The basic top-down algorithm is extended in the following respects.

First, in order to determine the positive examples E^+_{Rule} covered by the generated rule $Rule$ (procedure TestCoverage in figure 3), an abductive derivation is started for each positive example. This derivation results in a (possibly empty) set of abduced literals. We give as input to the abductive procedure also the set of literals abduced in the derivation of the previously covered examples. In this way, we ensure that the assumptions made during the derivation of the current example are consistent with the assumptions previously raised for covering other examples.

Second, in order to check that no negative example is covered ($E^-_{Rule} = \emptyset$ in figure 2) by the generated rule $Rule$, an abductive derivation is started for the default negation of each negative example ($\leftarrow not\_e^-$). Also in this case, each derivation does not start with an empty set of abducibles, but it starts from the set of abducibles previously assumed. The set of abducibles is initialized to the empty set at the beginning of the computation and is gradually extended as it is passed on from derivation to derivation. This is done throughout the whole learning process.

Third, after the generation of each clause, the abduced literals of target predicates are added to the training set, so that they become new training examples. For each positive abduced literal of the form $abd(\vec{c})$, the new positive example $abd(\vec{c})$ is added to E^+ set, while for each negative literals of the form $not\_abd(\vec{c})$ the negative example $abd(\vec{c})$ is added to E^-.

In order to be able to learn exceptions to rules, we have to include a number of predicates of the form not_abnorm_i in the bias for the target predicates. In this way, when the current partial rule in the specialization loop still covers some negative examples and no other literal can be added that would make it consistent, the rule is specialized by adding the literal $not\_abnorm_i(\vec{X})$ to its body. The negative examples previously covered are ruled out by abducing for them facts of the form $abnorm_i(e^-)$, while the positive examples will be covered by abducing the facts $not\_abnorm_i(e^+)$. These facts will then be added to the training set and will allow to learn rules for $abnorm_i(X)$, thus resulting in a definition for the exceptions to the current rule.

In order to learn a definition for the exceptions, the predicates $abnorm_i$ have to be considered as target predicates, and we have to define a bias for

them. Since we may have exceptions to exceptions, we may also include one[1] predicate of the form $not\_abnorm_j(\vec{X})$ in the bias for $abnorm_i$. Finally, for each couple of abducible predicates $abnorm_i, not\_abnorm_i$ we have to add to the background knowledge the constraint
$\leftarrow abnorm_i(\vec{X}), not\_abnorm_i(\vec{X})$.

It is worth mentioning that the treatment of exceptions by means of the addition of a non-abnormality literal to each rule is similar to the approach for declarative debugging followed in [14]. In order to debug a logic program, they first transform it by adding a different default literal to each rule. These literals are then used as assumptions for the correctness of the rule, to be possibly revised in the face of a wrong solution. The debugging algorithm identifies the assumptions that lead to the wrong solution, thus identifying the incorrect rules.

4 Example

In this section, we show the behaviour of the system in the case of learning exceptions to classification rules. The example is taken from [6].

Let us consider the following background abductive theory $B = \langle P, A, IC \rangle$ and training sets E^+ and E^-:

$$
\begin{aligned}
P = \ & \{bird(X) \leftarrow penguin(X). \\
& penguin(X) \leftarrow superpenguin(X). \\
& bird(a). \quad bird(b). \quad penguin(c). \quad penguin(d). \\
& superpenguin(e). \quad superpenguin(f).\} \\
A = \ & \{abnorm_1, abnorm_2, not\_abnorm_1, not\_abnorm_2\} \\
IC = \ & \{\leftarrow abnorm_1(X), not\_abnorm_1(X). \\
& \leftarrow abnorm_2(X), not\_abnorm_2(X).\} \\
& \leftarrow flies(X), not\_flies(X).\}
\end{aligned}
$$

$$
\begin{aligned}
E^+ &= \{flies(a), flies(b), flies(e), flies(f)\} \\
E^- &= \{flies(c), flies(d)\}
\end{aligned}
$$

Moreover, let the bias be[2]:

[1] More than one if we have different exceptions to different rules.
[2] For the sake of simplicity we avoid mentioning the abnormality predicates in the bias.

$flies(X) \leftarrow \alpha$ where $\alpha \subset \{superpenguin(X), penguin(X), bird(X),$
$not\_abnorm_1(X), not\_abnorm_2(X)\}$
$abnorm_1(X) \leftarrow \beta$ and $abnorm_2(X) \leftarrow \beta$ where
$\beta \subset \{superpenguin(X), penguin(X), bird(X)\}$

The algorithm first generates the following rule (R_1):
$flies(X) \leftarrow superpenguin(X).$
which covers $flies(e)$ and $flies(f)$ that are removed from E^+. Then, in the specialization loop, the rule $flies(X) \leftarrow bird(X)$. ($R_2$) is generated which covers all the remaining positive examples, but also the negative ones. In fact, the abductive derivations for $not\_flies(c)$ and $not\_flies(d)$ fail. Therefore, the rule must be further specialized by adding a new literal. The abducible literal not_abnorm_1 is added to the body of R_2 obtaining R_3:
$flies(X) \leftarrow bird(X), not\_abnorm_1(X).$
Now, the abductive derivations for $not\_flies(c)$ and $not\_flies(d)$ succeed abducing $\{abnorm_1(c), abnorm_1(d)\}$ while the derivations of the positive examples $flies(a)$ and $flies(b)$ succeed abducing
$\{not\_abnorm_1(a), not\_abnorm_1(b)\}$.
At this point, the system adds the abduced literals to the training set and tries to generalize them, by generating a rule for $abnorm_1$. The positive abduced literals for $abnorm_1$ form the set E^+, while the negative abduced literals form the set E^-. The resulting induced rule is (R_4):
$abnorm_1(X) \leftarrow penguin(X).$
No positive examples are now left in the training set therefore the algorithm ends by producing the following abductive rules:
$flies(X) \leftarrow superpenguin(X). \quad flies(X) \leftarrow bird(X), not\_abnorm_1(X).$
$abnorm_1(X) \leftarrow penguin(X).$
A result similar to ours is obtained in [6], but exploiting "classical" negation and priority relations between rules rather than abduction. By integrating induction and abduction, we can achieve greater generality with respect to [6].

5 Conclusions and Future Work

We have presented the system LAP for learning abductive logic programs. We consider an extended ILP problem in which both the background and target theory are abductive theories and coverage by deduction is replaced with coverage by abduction. The target theory can contain new rules containing abducible predicates.

LAP is obtained from the basic top-down algorithm of ILP by substituting the coverage of examples using deduction with the coverage using an abductive proof procedure. LAP has been implemented in Sicstus Prolog 3#3 and is able to learn abductive rules and rules with exceptions.

In the future, we will extend the system in order to be able to learn full abductive logic programs, comprehending also new integrity constraints. The integration of the algorithm with other systems for learning constraints, proposed in [10], seems very promising in this respect.

References

[1] H. Adé and M. Denecker. AILP: Abductive inductive logic programming. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995.

[2] M. Bain and S. Muggleton. *Non-Monotonic Learning*, chapter 7. Academic Press, 1992.

[3] F. Bergadano and D. Gunetti. *Inductive Logic Programming.* MIT press, 1996.

[4] F. Bergadano and D. Gunetti. Learning Logic Programs with Negation as Failure. In *Advances in Inductive Logic Programming.* IOS Press, 1996.

[5] M. Denecker, L. De Raedt, P. Flach, and A. Kakas, editors. *Proceedings of ECAI96 Workshop on Abductive and Inductive Reasoning.* Catholic University of Leuven, 1996.

[6] Y. Dimopoulos and A. Kakas. Learning Non-monotonic Logic Programs: Learning Exceptions. In *Proceedings of the 8th European Conference on Machine Learning*, 1995.

[7] Y. Dimopoulos and A. Kakas. Abduction and learning. In *Advances in Inductive Logic Programming.* IOS Press, 1996.

[8] F. Esposito, E. Lamma, D. Malerba, P. Mello, M. Milano, F. Riguzzi, and G. Semeraro. Learning abductive logic programs. In Denecker et al. [5].

[9] 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.

[10] A.C. Kakas and F. Riguzzi. Learning with abduction. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, 1997.

[11] E. Lamma, P. Mello, M. Milano, and F. Riguzzi. Integrating Induction and Abduction in Logic Programming. In P. P. Wang, editor, *Prooceedings of the Third Joint Conference on Information Sciences*, volume 2, pages 203–206, 1997.

[12] E. Lamma, P. Mello, M. Milano, and F. Riguzzi. Introducing Abduction into (Extensional) Inductive Logic Programming Systems. In *Proceedings of the 5th Congress of the Italian Association for Artificial Intelligence*, 1997.

[13] L. Martin and C. Vrain. A three-valued framework for the induction of general logic programs. In *Advances in Inductive Logic Programming*. IOS Press, 1996.

[14] L. M. Pereira, C. V. Viegas, and J. J. Alferes. Diagnosis and debugging as contradiction removal. In L. M. Pereira and A. Nerode, editors, *Proceedings of the 2nd International Workshop on Logic Programming and Non-monotonic Reasoning*, pages 316–330. MIT Press, 1993.