Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
email : Luc.DeRaedt@cs.kuleuven.ac.be

Attribute-Value Learning versus Inductive Logic Programming : the Missing Links (Extended Abstract)

Luc De Raedt

April 8, 1999

Abstract

Two contributions are sketched. A first contribution shows that a special case of relational learning can be transformed into attribute-value learning. However, it is much more tractable to stick to the relational representation than to apply the sketched transformation. This provides a sound theoretical justification for inductive logic programming. In a second contribution, we show how existing attribute-value learning techniques and systems can be upgraded towards inductive logic programming using the 'Leuven' methodology and illustrate it using the Claudien, Tilde, ICL, Warmr, TIC, MacCent and RRL systems.

1 Introduction

Relational learning and inductive logic programming have received a lot of attention recently, cf. [16, 5]. Also, the field has matured and has been applied to several important practical problems (cf. [3, 18] ...). Despite these successes, some fundamental questions about relational learning and inductive logic programming remain essentially unanswered. The first question concerns the need for relational learning. Though there exist now several impressive applications of relational learning (see e.g. [13]) and inductive logic programming and it is hard to see how these could have been realized with classical attribute value learning techniques, a precise argumentation is often lacking or implicit. The second question concerns the relation among relational learning and attribute value learning. Advocates of attribute-value learning often claim (in corridor discussions) that relational learning can be reduced to attribute-value learning, and hence that inductive logic programming is not (really) needed. Some of these arguments have their roots in database theory, but this author found no track of a precise and general argument in the literature (though many specific results are given in computational learning theory). Furthermore, (as argued

in this abstract) the typical argumentation based on the concept of the universal relation is insufficient. It turns out that - under certain (very restrictive) assumptions - inductive logic programming can be reduced to attribute-value learning. However, the transformation is computationally too expensive to be applied on real problems. Thus it is much more effective to stay with the relational representation and apply inductive logic programming systems. This provides a sound theoretical justification for the field of inductive logic programming.

If one is convinced of the use of inductive logic programming, one should also be interested in the development of inductive logic programming techniques and systems. To this aim, we briefly outline the 'Leuven' methodology for upgrading attribute value-learning systems towards using a first order representation. This methodology has its origin in the work by [8] on upgrading Valiant's PAC-learning results for k-CNF towards first order jk-clausal theories. Since then the same techniques has been applied to a number of systems including : the clausal discovery engine Claudien [7], the ICL system for classification [9], the TILDE system for top-down induction of logical decision trees [1], the Warmr system for first order association rules [11], the MacCent system based on the maximum entropy principle [10], the TIC system for top down induction of clustering [2], relational reinforcement learning [14], ...

2 A theoretical justification for ILP

In this abstract, we will use the learning from interpretations approach to inductive logic programming [8, 6]. More specifically, each example corresponds to a set of ground facts (or in relational database terminology, a small relational database in the form of a set of tuples) and are classified into a finite number of classes. E.g. one well-known Bongard problem can be represented as follows :

$$\{object(o1, triangle, red), object(o2, circle, blue), in(o1, o2)\}$$

This example contains a red triangle (referred to as o1) that is inside a blue circle (called o2). Though this example is a toy-problem its structure is very similar to that of well-known applications such as mutagenicity [18]. To represent the same information in a single relational database, one needs to add to each example a unique key which is then used by all relations and predicates.

We will also assume that examples are classified into a finite number of classes. Each class then corresponds to a propositional predicate, e.g. *positive*. The goal of the learning task is then to induce clauses to define these predicates. Such a definition will consist of a set of clauses of the following form :

$$class \leftarrow l_1, \dots, l_n$$

where the l_i are different atoms. E.g. a solution to the bongard problem could

be:

$$positive \leftarrow object(O1, triangle, X), object(O2, circle, Y), in(O1, O2)$$

To classify an example given a hypothesis H one has to assert H and the example in a Prolog database and run the query $?-positive$. If it succeeds, it is positive, otherwise, it is negative.

This framework corresponds to a simplification of the learning from interpretations setting in inductive logic programming, which was introduced by [8], and which is used in several inductive logic programming systems such as TILDE, ICL and Claudien. The above framework is also the simplest framework possible for relational learning and inductive logic programming, because classes are propositional, it avoids (and ignores) the difficulties with structured terms, recursion, background knowledge, continuous values and also because it can - under certain conditions - be reduced to the harder (and more popular) learning from entailment setting due to Gordon Plotkin (see [6] for more details). However, it is sufficient for our purposes. If we can justify this simple setting by contrasting it with attribute-value learning, it follows that generalizations of our setting are also justified w.r.t. attribute value learning.

2.1 Parameters of the learning task

We will distinguish the following parameters of the learning task.

Database parameters :

- r is the number of relations (excluding the predicate *positive*)
- i is the maximum number of tuples (facts) an example has in any single relation (or predicate)
- a is the maximum arity of a relation
- d is the maximum number of values of a given attribute
- e is the number of examples

In the above example for a Bongard type problem, $r = 2$, $i = 2$, and $a = 3$. Furthermore, the complexity of the learning task will also depend on the type of clauses and hypotheses searched for. We assume the following normal form for clauses.

$$class : -R_1(t_{1,1}, \dots, t_{1,n_1}), \dots, R_T(t_{T,1}, \dots, t_{T,n_T}), V_1 = U_1, \dots, V_J = U_J$$

where the $t_{i,j}$ are either constants or variables, such that each variable $t_{i,j}$ occurs exactly once in $R_1(t_{1,1}, \dots, t_{1,n_1}), \dots, R_T(t_{T,1}, \dots, t_{T,n_T})$, all V_i, U_i are different variables and V_i and U_i appearing in relations R . This type of clause can easily be translated into relational algebra (used in databases). The different literals

correspond to cartesian products and the equality literals correspond to possible join operations. E.g. the above clause translates to :

$positive \leftarrow object(O1, triangle, X), object(O2, circle, Y), in(O3, O4), O1 = O3, O2 = O4$

So, $T = 3$, and $J = 2$. Here T corresponds to the number of tuple (or relation) variables, and J to the *worst-case* number of joins. Hypothesis parameters :

- T is the maximum number of tuple variables in a clause of the hypothesis
- J is the maximum number of literals of type $V_i = U_i$ in a clause
- C is the maximum number of rules in a hypothesis.

Though from a theoretical perspective, some inductive logic programming or relational learning systems may search an infinite clause-space, from a practical perspective the search is often restricted to a finite space by restricting the parameters T and J , or variants thereof.

Given a learning problem in the above framework, there are two important questions that characterize the difficulty of the learning task. The first question concerns the complexity of the coverage test. Clearly, the more time it takes to test whether an example is covered by a hypothesis, the more difficult the learning task will be. An adequate measure for our purposes will be the complexity of testing whether a single clause covers an example. The second question concerns the complexity of the search-space. The more hypotheses it contains, the more difficult the learning task will be (generally speaking). An adequate measure for this will be the number of possible clauses allowed in hypothesis¹ Therefore we will be especially interested in the complexity of the following issues :

- Data Complexity : size of the data-set DC
- Query Complexity : complexity of testing whether a clause rule covers an example : QC
- Number of different rules in hypothesis language : HR
- Number of different hypotheses : H

It is possible to derive some rough estimates of these quantities (full details and proofs will be given in a longer version of this paper).

- $DC = O(e \cdot i \cdot a \cdot r)$
- Let $M = \min(J + 1, T)$, then $QC = O((i^M \cdot a \cdot M) + i \cdot a \cdot (T - M))$
- $HR = O(r^T \cdot (d + 1)^{aT} \cdot (aT)^{2J})$

It turns out that the query complexity is exponential in J and that the hypothesis space complexity is exponential in a , J and T . The size of the data set is less of a problem.

¹For the sake of simplicity, we will not use more complicated measures such as the VC-dimension which is common in computational learning theory.

2.2 Special cases

We now consider several special cases of the framework and discuss the relationship among them. All special cases can be obtained by instantiating some of the parameters of the given framework. Corresponding complexity results can be derived from Lemma 4.

2.2.1 Attribute value learning

Attribute value learning is the special case of relational learning where $i = 1, r = 1, T = 1, J = 0$.

2.2.2 Multi-instance learning

Dietterich et al. [12] consider the so-called multi-instance problem, which corresponds to : $r = 1, T = 1, J = 0$.² It is easily seen that the number of possible clauses remains unchanged with regard to attribute-value learning. Only the query complexity gets worse by a linear factor i .

It should also be noted that Dietterich et al. view the multi-instance problem as an important extension to the classical attribute-value learning approach. From the framework presented above, it follows that the multi-instance problem can also be regarded as a special case of inductive logic programming. Furthermore, it turns out that several of the topics for future work listed by Dietterich et al. have already been addressed by the inductive logic programming community. For instance, the design of extensions of well-known attribute-value learning systems such as decision trees to the multi-instance problem. Solutions to the multi-instance problem can often be obtained by setting specific (declarative) biases within existing inductive logic programming systems.

On the other hand, it will turn out that the multi-instance problem is at the heart of the difference between attribute-value learning and inductive logic programming. All inductive logic programming problems can easily be transformed into a multi-instance problem, but it is harder to transform the multi-instance problem in a classical attribute-value learning problem.

This touches already at the fundamental question of the relation among multi-instance learning and proper attribute value learning. More specifically, can we transform any multi-instance problem into a corresponding attribute-value learning problem ? The answer to this question is yes. The transformation will contain one (binary) attribute for each rule in the hypotheses space (bounded by the parameters J and T). There will then be one value for each example and attribute that basically expresses whether the clause covers the example or not. Given this transformation, a disjunctive attribute-value learning

²In the applications and algorithms of Dietterich et al. hypotheses are further biased by setting $C = 1$.

algorithm can be applied ³.

From a practical perspective, the above transformation is to be avoided because :

- the data-set grows too large.
- even if the attribute-value table would not be generated explicitly, any algorithm working with the attribute-value representation will degenerate into a generate-and-test algorithm at the level of clauses.

On the other hand, it seems hard to find alternative transformations that would not suffer from this problem. Furthermore, it is unnecessary to search for such transformations as it is easy to adapt existing the classical concept-learning algorithms to address this problem. This follows from the observation that the structure on the rule-space does not change, but only the coverage relation changes. Hence, no changes to the search strategy are need, only the point where coverage is tested should be modified in a straightforward manner.

2.2.3 Multi-tuple problems

The multi-tuple problem is a relaxation of the multi-instance problem where it is no longer required that $T = 1$, i.e. in multi-tuple problems $J = 0$, $r = 1$.

For the multi-tuple problems, coverage testing grows linearly and the search space exponentially in T as compared to the multi-instance problem.

Multi-tuple problems can easily be transformed into multi-instance problems (and hence also into attribute value format) by computing for each example the cartesian product of the relation with itself (T times). This transformation is again to be avoided because the data-set grows exponentially. A further complication is that it introduces syntactic variants in the hypothesis space. Again, the conclusion is that working with the original representation is more efficient.

2.2.4 Multi-join

The multi-join problem is a further relaxation of the multi-tuple problem where it is no longer required that $J = 0$.

Again it is possible to reduce the multi-join problem to the multi-instance problem (and hence to attribute value learning). The reduction involves adding a number of binary attributes to the transformation of the multi-tuple problem. Each binary attribute tests whether two variables are identical.

This transformation inherits its problems from the multi-tuple case.

³Disjunctive concepts in this context are of the form $A_1 = true \vee \dots A_n = true$ where the A_i are attributes.

2.2.5 Multi-relation ($r > 1$)

Using the universal relation concept from database theory, any relational learning problem can be transformed into a multi-join learning problem (with a single relation).

The transformation works as follows : Let $R_1, \dots, R_r$ be all the relations in the original learning problem (excluding *class*). Then compute for each example e the cartesian product of the R'_i , where $R'_i = R_i$ if there are tuples for R_i in e , otherwise $R'_i = (nil, \dots, nil)$. It is straightforward to see that the universal relation computed in this way is again exponentially larger than the original one. Therefore it is to be avoided.

Furthermore, notice also that this last transformation *alone* is insufficient to transform an inductive logic programming problem into attribute-value form.

2.3 Conclusions

We have shown that the simplest form of inductive logic programming (or relational learning) can be transformed into an attribute-value form. However, we have also argued that each transformation results in computational problems. Therefore, it is more effective to stick to the original representation. Technically speaking, it seems that the multi-instance problem forms the natural frontier between inductive logic programming and classical attribute-value learning.

3 Upgrading attribute-value learning systems

Given that one is confronted with a true relational learning problem, the question is how to develop learning algorithms for solving it. In recent years, the 'Leuven' team has developed a number of novel inductive logic programming systems using a single methodology.

The methodology is derived from the PAC-learning result for jk -clausal theories by [8]. This result proposes to use Herbrand interpretations (i.e. sets of ground facts as specified above) as the representation of examples. It directly upgrades Valiant's seminal result for k -CNF [19]. This property is important because it means that the original propositional result is obtained as a special case of that for first order logic. Reversing this property is the basis of the methodology: inductive logic programming systems can be obtained by starting from propositional ones and changing the representation from feature-vectors to Herbrand interpretations.

This results in essentially the following 'Leuven' methodology :

1. start from a propositional learning technique and algorithm that matches the problem closest,
2. modify the problem setting of that algorithm as to use examples that are first order (Herbrand) interpretations instead of attribute-value vectors,

3. modify the representation of the hypotheses in the propositional case to a form close to clausal logic (e.g. for the TILDE system, a first order decision tree representation has been developed),
4. modify the generalization and specialization operators of the propositional systems as to use the well-known θ -subsumption framework for generalization⁴,
5. retain all other features of the propositional algorithm if possible (this includes : search, heuristics, noise-handling, pruning, ...),
6. implement and evaluate the first order technique.

This methodology has resulted in a number of novel inductive logic programming systems. More specifically, Claudien [7] and the more recent Warmr [11] upgrade the association rule paradigm of data mining, Tilde [1] upgrades Quinlan's C4.5 algorithm [17], ICL [9] upgrades CN2 [4], MacCENT [10] upgrades propositional maximum entropy approaches, and relational reinforcement learning upgrades Q-learning as described by [15].

A more detailed case study with this methodology is in preparation (in collaboration with Wim Van Laer).

Acknowledgements

This work is supported by the Fund for Scientific Research, Flanders and the ESPRIT project no. 20237 on Inductive Logic Programming. I'd like to thank the members of the 'Leuven' team for their contributions and a long list of other persons for discussions (cf. the longer versions of this paper).

From Luc.DeRaedt@cs.kuleuven.ac.be Thu Apr 8 19:14:10 1999 Return-Path: jLuc.DeRaedt@cs.kuleuven.ac.be Received: from mailgateway1.uni-freiburg.de (mailgateway1.uni-freiburg.de [132.230.1.6]) by avalon.informatik.uni-freiburg.de (8.9.0/8.9.0) with SMTP id TAA03034 for jderaedt@informatik.uni-freiburg.de; Thu, 8 Apr 1999 19:14:09 +0200 (MET DST) Received: from styx.cs.kuleuven.ac.be [134.58.40.3] (root) by mailgateway1.uni-freiburg.de with esmtp (Exim 1.82 3) id 10VINT-0003pz-00; Thu, 8 Apr 1999 19:14:07 +0200 Received: from droopy.cs.kuleuven.ac.be (lucdr@droopy.cs.kuleuven.ac.be [134.58.41.10]) by styx.cs.kuleuven.ac.be (8.9.1a/8.9.1) with ESMTP id TAA28279 for jderaedt@informatik.uni-freiburg.de; Thu, 8 Apr 1999 19:10:58 +0200 (MET DST) Received: (from lucdr@localhost) by droopy.cs.kuleuven.ac.be (8.9.1a/8.9.1) id TAA12780 for deraedt@informatik.uni-freiburg.de; Thu, 8 Apr 1999 19:10:57 +0200 (MET DST) Date: Thu, 8 Apr 1999 19:10:57 +0200 (MET DST) From: Luc De Raedt jLuc.DeRaedt@cs.kuleuven.ac.be; Message-Id: j199904081710.TAA12780@droopy.cs.kuleuven.ac.be; Content-Type: text X-Mozilla-Status: 0001 Content-Length: 3953

⁴ θ -subsumption is sufficient for most purposes in inductive logic programming; problems only arise with recursion or multiple-predicate learning.

References

- [1] H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 1998. To appear.
- [2] H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees. In *Proceedings of the 15th International Conference on Machine Learning*, 1998.
- [3] I. Bratko and S. Muggleton. Applications of inductive logic programming. *Communications of the ACM*, 38(11):65–70, 1995.
- [4] P. Clark and T. Niblett. The CN2 algorithm. *Machine Learning*, 3(4):261–284, 1989.
- [5] L. De Raedt, editor. *Advances in Inductive Logic Programming*, volume 32 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 1996.
- [6] L. De Raedt. Logical settings for concept learning. *Artificial Intelligence*, 95:187–201, 1997.
- [7] L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.
- [8] L. De Raedt and S. Džeroski. First order jk -clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.
- [9] L. De Raedt and W. Van Laer. Inductive constraint logic. In *Proceedings of the 5th Workshop on Algorithmic Learning Theory*, volume 997 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1995.
- [10] L. Dehaspe. Maximum entropy modeling with clausal constraints. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 109–124. Springer-Verlag, 1997.
- [11] L. Dehaspe and L. De Raedt. Mining association rules in multiple relations. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 125–132. Springer-Verlag, 1997.
- [12] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.
- [13] S. Džeroski and I. Bratko. Applications of inductive logic programming. In L. De Raedt, editor, *Advances in inductive logic programming*, volume 32 of *Frontiers in Artificial Intelligence and Applications*, pages 65–81. IOS Press, 1996.

- [14] S. Džeroski, L. De Raedt, and H. Blockeel. Relational reinforcement learning. In *Proceedings of the International Conference on Machine Learning*. Morgan Kaufmann, 1998.
- [15] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [16] S. Muggleton and L. De Raedt. Inductive logic programming : Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.
- [17] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in machine learning. Morgan Kaufmann, 1993.
- [18] A. Srinivasan, S.H. Muggleton, M.J.E. Sternberg, and R.D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85, 1996.
- [19] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.