Lookahead and Discretization in ILP

Hendrik Blockeel and Luc De Raedt

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A
3001 Heverlee
e-mail: {Hendrik.Blockeel, Luc.DeRaedt}@cs.kuleuven.ac.be

Abstract. We present and evaluate two methods for improving the performance of ILP systems. One of them is discretization of numerical attributes, based on Fayyad and Irani's text [9], but adapted and extended in such a way that it can cope with some aspects of discretization that only occur in relational learning problems (when indeterminate literals occur). The second technique is lookahead. It is a well-known problem in ILP that a learner cannot always assess the quality of a refinement without knowing which refinements will be enabled afterwards, i.e. without looking ahead in the refinement lattice. We present a simple method for specifying when lookahead is to be used, and what kind of lookahead is interesting. Both the discretization and lookahead techniques are evaluated experimentally. The results show that both techniques improve the quality of the induced theory, while computational costs are acceptable.

1 Introduction

Propositional learning has been studied much more extensively than inductive logic programming (ILP), and at this moment the former field is better understood than the latter. However, ILP shares many techniques, heuristics etc. with propositional learning, and therefore it can often profit from results obtained for propositional learners. Due to several aspects of ILP that do not occur in propositional learning, it is often necessary to adapt these techniques to the specific ILP context.

In this paper, we discuss two such upgrades of propositional learning results to the ILP context. The first is discretization of continuous attributes. Irani and Fayyad [9] have presented a propositional method that divides a continuous domain into several subsets which can then be used as discrete values. We will briefly discuss the method, show that ILP poses some problems with respect to discretization that do not occur in propositional learning, and propose an adaptation of the method.

The second topic we shall discuss, is the use of lookahead. It is a well-known problem with relational learners that most heuristics (e.g. information gain) have problems with assessing the quality of a refinement of a rule, because a single literal that is added by the refinement step may not cause any gain, but may be very important to make the addition of gainful literals possible later on (because

it introduces new variables). The advantage of adding a literal may only become clear further down the refinement tree. In general, heuristics that work well for propositional learners do not always perform as well for relational learners.

We propose a lookahead technique to alleviate this problem. By allowing the learner to look more than one level ahead in the refinement lattice, it may be able to assess the quality of a refinement more accurately.

Both the discretization and lookahead methods have been implemented in a novel ILP system called TILDE, and we present experimental results confirming the usefulness of both techniques.

This text is organized as follows. In Section 2, we briefly discuss the ILP setting that is used. In Section 3 we discuss discretization, in Section 4 lookahead. Conclusions are presented in Section 5.

2 The Learning Setting

We essentially use the *learning from interpretations* paradigm for inductive logic programming, introduced by [4], and related to other inductive logic programming settings in [3].

In this paradigm, each example is a Prolog knowledge base (i.e. a set of definite clauses), encoding the specific properties of the example. Furthermore, each example is classified into one of a finite set of possible classes. One may also specify background knowledge B in the form of a Prolog knowledge base.

More formally, the problem specification is :

Given: a set of classes C, a set of classified examples E, and a background theory B,
Find: a hypothesis H (a set of definite clauses in Prolog), such that for all $e \in E$: $H \wedge e \wedge B \models c$, and $H \wedge e \wedge B \not\models c'$ where c is the class of the example e and $c' \in C - \{c\}$.
Our experiments have been done with the ILP system TILDE[1], which represents the induced hypotheses as logical decision trees (these are a first order logic upgrade of the classical decision trees used in propositional concept learning).

Example 1. Suppose a number of machines are under revision. Some have to be sent back to the manufacturer, and others can be kept. The aim is to predict whether a machine needs to be sent back.

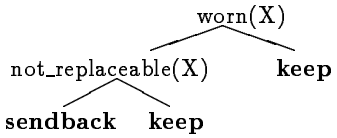
Given the following set of examples (each example represents one machine):

Example 1	**Example 2**	**Example 3**	**Example 4**
class(keep)	class(sendback)	class(sendback)	class(keep)
`worn(gear)`	`worn(engine)`	`worn(control_unit)`	`worn(chain)`
`worn(chain)`	`worn(chain)`		

and the following background knowledge:

Background knowledge
`replaceable(gear)`
`replaceable(chain)`
`not_replaceable(engine)`
`not_replaceable(control_unit)`

the following decision tree (in first order logic) represents a correct classification procedure:



We wish to stress that — although for our experiments we use only TILDE— the techniques, problems and solutions discussed in this paper generally apply to any ILP-learner.

3 Discretization

3.1 Principle

The motivation for discretizing numeric data is twofold and based on the findings in attribute value learning. On the one hand, there is an efficiency concern. On the other hand, by discretizing the data, one may sometimes obtain higher accuracy rates.

Most current ILP systems ([13, 12] generate numbers during the induction process itself, which may cause a lot of overhead: at each refinement step (a lot of) constants need to be generated. TILDE, however, discretizes numeric domains beforehand, which makes the induction process much more efficient. It is known that in a propositional learning context this does not necessarily decrease the quality of the induced hypothesis (it may even increase; see e.g. [2]).

In our approach to discretization, the user can declaratively identify the relevant queries and the variables for which the values are to be discretized. For instance, `to_be_discretized(atom(A,B,C,D), [D])` states that the fourth argument of *atom* should be discretized.

The resulting numeric attributes are then discretized using a simple modification of Fayyad and Irani's method. The details of this method can be found in [9] and [7]. In short, the algorithm finds a threshold that partitions a set of examples into two subsets such that the average class entropy of the subsets is as small as possible. This procedure is applied recursively on S_1 and S_2 until some stopping criterion is reached.

With respect to Fayyad and Irani's algorithm, two adaptations were made. Firstly, Fayyad and Irani propose a stopping criterion that is based on the minimal description length principle, but for TILDE we found this method to generate

very few thresholds. Therefore TILDE's discretization procedure accepts a maximal number of thresholds as a parameter. This has the additional advantage that one can experiment with different numbers of thresholds.

A second adaptation made to Fayyad and Irani's method specifically concerns non-determinacy. Due to the fact that one example may have multiple or no values for a numeric attribute, we use sum of weights instead of number of examples in the appropriate places of Fayyad and Irani's formulae (in the attribute value case all values have weight 1 as each example has only one value for one attribute). The sum of the weights of all values for one numeric attribute or query in one example always equals one, or zero when no values are given.

Example 2. Consider an example $E_1 = \{p(1), p(2), p(3)\}$, and some threshold $T = 2.5$. In a propositional context T would partition a set of examples S into S_1 (examples that have a value < 2.5) and S_2, the rest. In our context, E_1 has weight $2/3$ in S_1, and $1/3$ in S_2.

Aside from the generation of subintervals, there is the topic of how these intervals should be used. We see several possibilities:

- Using inequalities to compare whether a value is less than a discretization threshold; this corresponds to an *inequality* test in the discrete domain.
- Checking whether a value lies in some interval bounded by two consecutive thresholds. Such an interval test corresponds with an *equality* test in the discretized domain.
- Checking whether a value lies in an interval bounded by non-consecutive thresholds. This corresponds to an *interval* test in the discrete domain.

The three approaches have been used and compared in our experiments.

3.2 Experimental Evaluation

We evaluate the effect of discretization on two datasets: the Musk dataset (available at the UCI repository [11]) and the Diterpene dataset, generously provided to us by Steffen Schulze-Kremer and Sašo Džeroski. Both datasets contain non-determinate numerical data, which makes them fit to test our discretization procedure on. We refer to [5] and [8] for precise descriptions of the datasets.

On the Musk dataset, we have compared the discretization approaches using equality tests, and using inequality tests. On the Diterpene dataset, the interval and inequality approaches were compared with using no discretization at all. In Figure 1, theory accuracies are plotted against the maximal number of thresholds that was given. Figure 2 shows running times on the Diterpene dataset.

Our conclusions are that the way in which discretization results are used (discrete (in)equalities, intervals) significantly influences the accuracy of the induced theory, as well as the efficiency of the induction process. A good choice of the number of thresholds is also important. Accuracy often shows a clear trend of reaching a maximum at some number of thresholds, then slowly decreases. A good combination of thresholds and discretization method may increase both accuracy and efficiency.

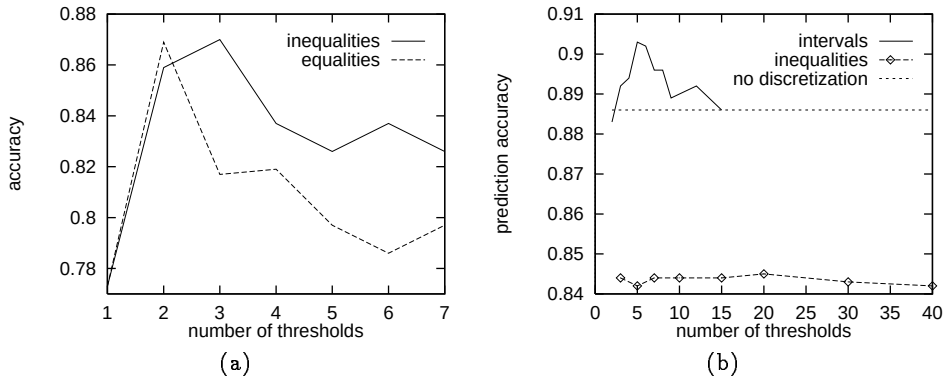


Fig. 1. Influence of number of thresholds on accuracy: (a) Musk dataset, comparing equalities and inequalities; (b) Diterpene dataset, comparing intervals with inequalities and no discretization at all

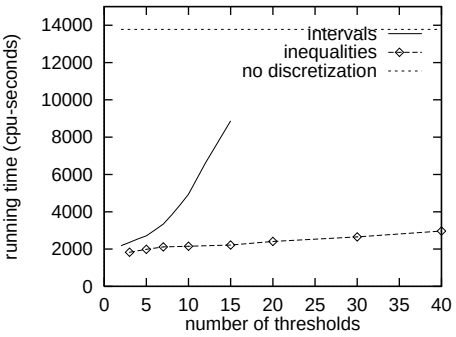


Fig. 2. Comparison of running times for the different approaches (Diterpene dataset)

4 Lookahead

4.1 Principle

An important problem in ILP is that refinement of a clause by adding a literal may result in little immediate improvement, although the literal may introduce new variables that are important for classification. For greedy systems, this may heavily influence the induction process. Although some systems have some provisions to cope with the problem (e.g. FOIL [13] automatically adds determinate literals), it is still an open question how it can best be solved.

A possible technique for coping with the problem, is to make the learner look ahead in the refinement lattice. When a literal is added, the quality of the refinement can better be assessed by looking at the additional refinements that will become available after this one, and looking at how good these are. This technique is computationally expensive, but may lead to significant improvements to the induced theories.

There are several ways in which lookahead can be performed. One is to look at further refinements in order to have a better estimate for the current refinement. In that case, the heuristic value assigned to a refinement c' of a clause c is a function of c' and $\rho(c')$, where ρ is a classical refinement operator under θ-subsumption. ρ itself does not change with this form of lookahead.

A second kind of lookahead is to redefine the refinement operator itself so that the two-step-refinements are incorporated in it. That is, if the original refinement operator (without lookahead) is ρ', then $\rho(c) = \rho'(c) \cup \{\rho'(c') | c' \in \rho'(c)\}$. This approach, as well as the former one, can be extended in the sense that the learner could look more than one level ahead.

The TILDE system follows the second approach. It relies on the user to provide some information about when lookahead is needed, because we believe that in many cases the user has a better idea about this than what a learning system can derive on the basis of e.g. determinacy. The user can provide templates of the form `lookahead(`C_1`,` C_2`)`, specifying that whenever a conjunction is added matching C_1, the conjunction C_2 may be added as well.

4.2 Experimental Evaluation

We have tested the effect of lookahead on two datasets: the Mutagenesis dataset and the Mesh dataset. These two were chosen because they are widely used as ILP benchmarks, and because they contain structural data where properties of neighbouring substructures (atoms or edges) are important for classification, but the link to a neighbour itself (bonds, *neighbour* predicate) provides little or no gain (therefore lookahead is important). We refer to [14] and [6] respectively for more information on these datasets.

Both datasets repeatedly were partitioned into 10 subsets. Two tenfold cross-validations were run based on each such partition; one without allowing lookahead, and one with lookahead. In Figure 3, each dot represents one partition; dots above the straight line are those partitions where accuracy with lookahead was higher than without lookahead. For both the Mutagenesis and Mesh datasets, improvements obtained by using lookahead are significant at the 1% level.

TILDE's performance is compared with FOIL's and Progol's on the Mutagenesis dataset in Table 1. It can be seen that, although lookahead is computationally expensive, it is still much cheeper than an exhaustive search (such as Progol performs).

	FOIL	Progol	TILDE, no lookahead	TILDE, lookahead
accuracy	61 %	76 %	74.6 %	77.0 %
time	4950 s	117039 s	23 s	539 s

Table 1. Results on Mutagenesis dataset

We conclude that in both cases the ability to use lookahead improves TILDE's

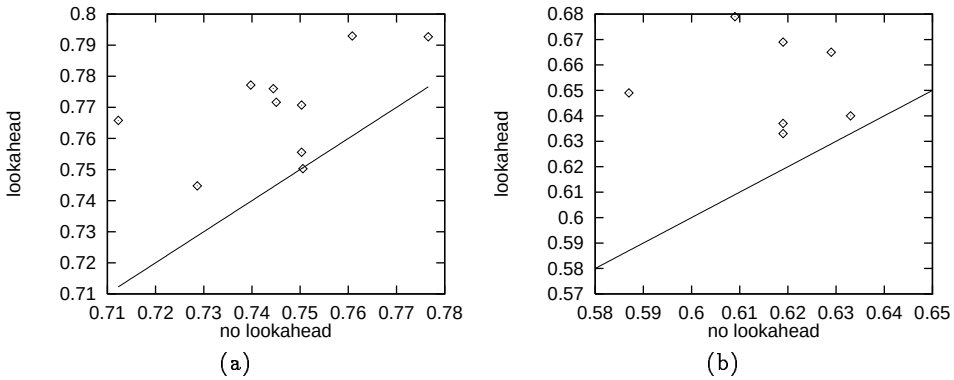


Fig. 3. Comparison of TILDE's performance with and without lookahead, (a) on the Mutagenesis data; (b) on the Mesh data

performance significantly. Computational complexity also increases, but is still acceptable.

5 Conclusions and Related Work

We have presented two methods for improving the performance of ILP-systems: discretization, as an extended version of propositional discretization; and lookahead, which in itself is an feature that can be added to any greedy search technique. We have evaluated these methods experimentally. Conclusions are that both techniques can lead to higher accuracy. Discretization also increases efficiency, but lookahead has a negative effect on this. An interesting observation is that the way in which discretization results are used (inequality tests or interval tests) can have a significant impact on prediction accuracy.

Although our experiments only concern TILDE, the proposed techniques are generally applicable; other ILP systems might profit from them as well.

Our work has of course heavily been influenced by several publications on discretization ([9, 7]) and especially [15]. The idea of using lookahead in ILP has been uttered several times before (e.g. [10]).

Acknowledgements

Hendrik Blockeel is supported by the Flemish Institute for the Promotion of Scientific and Technological Research in Industry (IWT). Luc De Raedt is supported by the Fund for Scientific Research of Flanders. This work is also part of the European Community Esprit project no. 20237, ILP2.

The authors thank Steffen Schulze-Kremer, the Max-Planck Institute for Molecular Genetics, Berlin, and Sašo Džeroski, for providing the Diterpene dataset; and Wim Van Laer and Sašo Džeroski for previous research on discretization for ILP systems.

References

1. H. Blockeel and L. De Raedt. Experiments with top-down induction of logical decision trees. Technical Report CW 247, Dept. of Computer Science, K.U.Leuven, January 1997. Also in Periodic Progress Report ESPRIT Project ILP2, January 1997. `http://www.cs.kuleuven.ac.be/publicaties/rapporten/CW1997.html`.
2. J. Catlett. On changing continuous attributes into ordered discrete attributes. In Yves Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 164–178. Springer-Verlag, 1991.
3. L. De Raedt. Induction in logic. In R.S. Michalski and Wnek J., editors, *Proceedings of the 3rd International Workshop on Multistrategy Learning*, pages 29–38, 1996.
4. L. De Raedt and S. Džeroski. First order *jk*-clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.
5. 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.
6. B. Dolšak and S. Muggleton. The application of Inductive Logic Programming to finite element mesh design. In S. Muggleton, editor, *Inductive logic programming*, pages 453–472. Academic Press, 1992.
7. J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In A. Prieditis and S. Russell, editors, *Proc. Twelfth International Conference on Machine Learning*. Morgan Kaufmann, 1995.
8. S. Džeroski, S. Schulze-Kremer, K. R. Heidtke, K. Siems, and D. Wettschereck. Applying ILP to diterpene structure elucidation from 13C NMR spectra. In *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 14–27, August 1996.
9. U.M. Fayyad and K.B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1022–1027, San Mateo, CA, 1993. Morgan Kaufmann.
10. N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
11. C.J. Merz and P.M. Murphy. UCI repository of machine learning databases [`http://www.ics.uci.edu/~mlearn/mlrepository.html`], 1996. Irvine, CA: University of California, Department of Information and Computer Science.
12. S. Muggleton. Inverse entailment and progol. *New Generation Computing*, 13, 1995.
13. J.R. Quinlan. FOIL: A midterm report. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1993.
14. A. Srinivasan, S.H. Muggleton, and R.D. King. Comparing the use of background knowledge by inductive logic programming systems. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, 1995.
15. W. Van Laer, S. Džeroski, and L. De Raedt. Multi-class problems and discretization in ICL (extended abstract). In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programming (ILP for KDD)*, 1996.

This article was processed using the LaTeX macro package with LLNCS style