On Multi-class Problems and Discretization in Inductive Logic Programming

Wim Van Laer[1], Luc De Raedt[1], Sašo Džeroski[2]

[1] Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
[2] Department of Intelligent Systems, Jožef Stefan Institute
Jamova 39, 1111 Ljubljana, Slovenia
Email:{WimV,LucDR}@cs.kuleuven.ac.be, saso.dzeroski@ijs.si

Abstract. In practical applications of machine learning and knowledge discovery, handling multi-class problems and real numbers are important issues. While attribute-value learners address these problems as a rule, very few ILP systems do so. The few ILP systems that handle real numbers mostly do so by trying out all real values applicable, thus running into efficiency or overfitting problems.
The ILP learner ICL (Inductive Constraint Logic), learns first order logic formulae from positive and negative examples. The main characteristic of ICL is its view on examples, which are seen as interpretations which are true or false for the target theory. The paper reports on the extensions of ICL to tackle multi-class problems and real numbers. We also discuss some issues on learning CNF formulae versus DNF formulae related to these extensions. Finally, we present experiments in the practical domains of predicting mutagenesis, finite element mesh design and predicting biodegradability of chemical compounds.

Keywords: Learning, Knowledge Discovery, Inductive Logic Programming, Classification, Discretization.

1 Introduction

The ILP system ICL (Inductive Constraint Logic, see [7]) does not employ the traditional ILP semantics in which examples are clauses that are (resp. are not) entailed by the target theory. It rather takes the view that examples are logical interpretations that are a model (resp. not a model) of the unknown target theory. This view originates from computational learning theory where it was originally applied to boolean concept-learning [18], but recently upgraded towards 1st order logic [6] where it is known as learning from interpretations [4].

The ICL system can be considered an upgrade of the attribute value learning system CN2 [3]. However, whereas CN2 learns boolean concepts in DNF form, ICL learns first order theories in CNF form.

In this paper, we import further features of attribute value learning into the ILP system ICL. First, it is shown that ICL can also learn DNF concepts. This is realized using a (logically sound) transformation on inputs and outputs. Secondly, CN2's way of handling multi-class problems (i.e. problems where examples can belong to more than 2 classes) is also incorporated within ICL. Thirdly, and most importantly, (as many other ILP systems) ICL has problems with handling real numbers. In attribute value learning, discretization has recently received a lot of attention (cf. [2, 9]) and proven to be a valuable technique. We show how Fayyad and Irani's discretization technique ([11, 9]) can be modified for use in ILP and more specifically in ICL.

The paper is structured as follows. In Section 2, some logical background is given and the framework in which ICL works is described. Section 3 discusses some issues on the CNF and DNF representations used in ICL. In Sections 4 and 5, we show how we can handle problems with more than two classes and data with real numbers. Section 6 describes some experiments that indicate the usefulness of the newly incorporated features in ICL. Section 7 concludes.

2 The Learning System ICL

An overview of ICL can be found in [7]. Here, we will describe the framework of ICL in an informal way and discuss some practical aspects.

But first we will introduce some concepts from logic (for an introduction to first order logic and model theory, we refer to [14, 12]).

A first order alphabet is a set of predicate symbols, constant symbols and functor symbols. An atom $p(t_1, ..., t_n)$ is a predicate symbol p followed by a bracketed n-tuple of terms t_i. A term t is a variable V or a function symbol f immediately followed by a bracketed n-tuple of terms t_i. Constants are function symbols of arity 0. A literal l is an atom or the negation of an atom. Atoms are positive literals, negated atoms are negative literals.

A **CNF** (Conjunctive Normal Form) expression is of the form
$(\forall V_{1,1}, ..., V_{1,v_1} : l_{1,1} \vee ... \vee l_{1,n_1}) \wedge ... \wedge (\forall V_{k,1}, ..., V_{k,v_k} : l_{k,1} \vee ... \vee l_{k,n_k})$
where $l_{i,j}$ are literals and $V_{i,1}, ..., V_{i,v_i}$ are variables occurring in $l_{i,1} \vee ... \vee l_{i,n_i}$.

A **DNF** (Disjunctive Normal Form) expression is of the form
$(\exists V_{1,1}, ..., V_{1,v_1} : l_{1,1} \wedge ... \wedge l_{1,n_1}) \vee ... \vee (\exists V_{k,1}, ..., V_{k,v_k} : l_{k,1} \wedge ... \wedge l_{k,n_k})$
where $l_{i,j}$ are literals and $V_{i,1}, ..., V_{i,v_i}$ are variables occurring in $l_{i,1} \wedge ... \wedge l_{i,n_i}$.
The symbol $\forall$ reads 'for all' and stands for universal quantification, and $\exists$ reads 'there exists' and stands for existential quantification. For instance, the formula $\exists C, T : triangle(T) \wedge circle(C) \wedge in(C, T)$ states that there exist a triangle and a circle such that the circle is inside the triangle.

A *Herbrand interpretation* over a first order alphabet is a set of ground atoms constructed with the predicate, constant and functor symbols in the alphabet. A Herbrand interpretation corresponds in the boolean or propositional case to a variable assignment. The meaning of a Herbrand interpretation is that all atoms in the interpretation are true, and all other atoms are false.

A *substitution* $\theta = \{V_1 \leftarrow t_1, ..., V_n \leftarrow t_n\}$ is an assignment of terms $t_1, ..., t_n$ to variables $V_1, ..., V_n$.

By now, we can define the *truth and falsity* of an *expression* in a Herbrand interpretation (if an expression is true in an interpretation we also say that the interpretation is a model for the expression):

A **ground literal** l is true in an interpretation I if and only if l is a positive literal and $l \in I$, or l is a negative literal and $l \notin I$.
A **CNF** expression (defined as above) is true in an interpretation if and only if for all i and for all substitutions θ such that $(l_{i,1} \vee ... \vee l_{i,n_i})\theta$ is ground, at least one of the $l_{i,j}\theta$ is true in I.
A **DNF** expression (defined as above) is true in an interpretation I if and only if there exists an i and a substitution θ such that $(l_{i,1} \wedge ... \wedge l_{i,n_i})\theta$ is ground, and all $l_{i,j}$ are true in I.
Let us illustrate this rather complicated definition. It states that e.g. $flies \vee \neg bird \vee \neg abnormal$ is true in the interpretations $\{flies\}, \{abnormal\}$ but false in $\{bird,\ abnormal\}$. Similarly, it allows us to say that $\exists C, T : triangle(T) \wedge circle(C) \wedge in(C,T)$ is true in $\{triangle(t), circle(c), in(c,t), large(c), small(t)\}$ and false in $\{triangle(t),\ circle(c)\}$. Furthermore, $\forall X : polygon(X) \vee \neg square(X)$ is true in $\{square(s), polygon(s)\}$ and in $\{circle(c)\}$ but false in $\{square(s)\}$.

2.1 The Framework of ICL

ICL is a classification system within the framework of *learning from interpretations*. In this framework, the following choices are made: examples are (Herbrand) interpretations and a hypothesis covers an example if the hypothesis is true in the interpretation. ICL can thus be described as follows:
Given a set of positive and negative examples, and a language $\mathcal{L}_{\mathcal{H}}$, find a first order theory $\mathcal{H} \subset \mathcal{L}_{\mathcal{H}}$ that is complete (covers all positive examples) and consistent (covers no negative examples).

At the moment, ICL can learn first order theories in Conjunctive Normal Form (CNF) and Disjunctive Normal Form (DNF) (see also section 3).

To illustrate the *learning from interpretations* paradigm and our ICL setting, we take one of the Bongard problems. These are general problems developed by the Russian scientist M. Bongard in his book *Pattern recognition.* Each problem consists of 12 figures, six of class $\oplus$ and six of class $\ominus$. One example problem can be found in Fig. 1. The goal is to discriminate between the two classes.

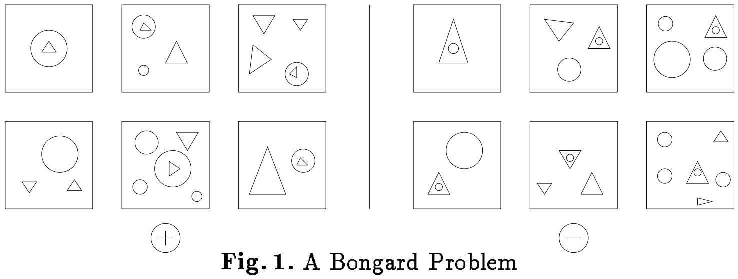

Fig. 1. A Bongard Problem

Each of these figures can be described by a set of facts. Take for instance the upper left example in Fig. 1. It consists of a small triangle, pointing up, which is in a large circle. This figure can be described as: $I = \{$*triangle(f1), small(f1), up(f1), circle(f2), large(f2), in(f1,f2)*$\}$. The following DNF theory is consistent and complete: $\exists X \exists Y : \mathit{triangle}(Y) \wedge \mathit{in}(Y, X)$. This theory says that for each figure of class $\oplus$ there exists a triangle that is inside another object.

2.2 Practice

Currently, ICL is implemented in ProLog by BIM. Several heuristics (based on CN2) are incorporated to handle noise. This means that the learned theory need not be strictly complete and consistent with the training set: a theory might not cover all positives, and not all negatives need to be excluded by the theory. ICL has several user-tunable parameters, such as the significance level for significance tests, the maximum number of literals in disjuncts/conjuncts in CNF/DNF, the beam size, and the search heuristic. More details on the algorithm (based on the covering approach of CN2 with unordered rules) and the heuristics can be found in [7].

To specify the hypothesis language, ICL uses the same declarative bias as Claudien, i.e. DLAB (declarative language bias, see [5]). DLAB is a formalism for specifying an intensional syntactic definition of the language $\mathcal{L}_{\mathcal{H}}$. For CNF, the hypothesis is a conjunction of clauses, and DLAB specifies the allowed syntax for the head and the body. For DNF, the hypothesis is a disjunction of rules (each rule being a conjunction of literals), and DLAB specifies the allowed syntax for the positive and negative literals. This is automatically translated into a refinement operator (under θ-subsumption) for the specified language which is used by ICL to traverse the search space. A small example:

```
{false <-- 0-len:[len-len:[lumo(Lumo), lt(Lumo, 1-1:[-1, -2])],
                  len-len:[atom(A1, Elem1, Type1, Charge1),
                           lt(Charge1, 1-1:[-0.2, -0.1, 0, 0.1])]]}
```

Min-Max:List means that at least Min and at most Max literals of List are allowed (`len` is the length of `List`). Note that lt(Lumo, 1-1:[-1,-2]) is a shorthand for 1-1:[lt(Lumo, -1), lt(Lumo, -2)].

3 CNF and DNF Representation

Originally, ICL learned a hypothesis in conjunctive normal form (CNF). This is inherited from its older twin system Claudien (see [5]).

In [15], R. Mooney presents 2 dual algorithms, one for learning CNF and one for learning DNF. However, it turns out that the CNF algorithm can be used to learn DNF formulae provided that the role of the positives and the negatives is swapped, the negation of the tests is used and the result is negated (this property was not mentioned in Mooney's paper). Thus, the CNF version of ICL can be used to learn DNF. For convenience, we have adapted the output procedure of ICL-CNF so that it can produce a DNF output.

Experiments indicate there is a difference in classification accuracy when learning CNF or DNF. In the mutagenesis case this is very clear. Table 1 shows the theory accuracy for the learned DNF and CNF theory. Theory complexity also differs. This is important in the light of Mooney's [15] experiments. He argued that the reason for the differences in accuracy and complexity can be explained by the difference in CNF and DNF. From the property above, it follows that there is an alternative explanation. The differences can be explained as well by the differences in learning the negative concept instead of the positive one.

4 Multi-class Problems

In the previous sections, ICL has been described as a learning system that can be applied to problems with 2 classes of examples. Given a set of positive and negative examples for a class c, ICL learns a theory T^c (in CNF $=(D_1 \wedge ... \wedge D_n)$, in DNF $= (C_1 \vee ... \vee C_n)$) that discriminates between the positive and negative examples. An unseen example is then classified as class c if the example is a model for the theory T^c (i.e. in CNF: all clauses/disjuncts D_i of the theory cover the example; in DNF: at least one conjunct C_i of the theory covers the example). Otherwise the example is assumed not to be of class c.

In many applications with two classes, this is sufficient. But if we have m classes, it's not sufficient to learn just one theory (one would only be able to discriminate between one class and all the others). We should learn m theories, one for each class. The question then is, how to apply this set of theories to an unseen example in order to predict its class? (This is also useful for problems with two classes, as errors in one theory can be undone by the other theory).

In ICL, we use a similar strategy as in CN2 (see [3]). Given a problem with m classes ($m \geq 2$), we first learn m separate DNF theories for the m different classes. When merging these theories $T_1 ... T_m$ into T_{multi}=$(C_{1,1} \vee ... \vee C_{1,n_1} \vee ... \vee C_{m,1} ... \vee C_{m,n_m})$, we store with each rule/conjunct $C_{i,j}$ in T_{multi}, the distribution of covered (training) examples among the classes (this is a vector $V_{i,j}$ of length m, where the kth element of $V_{i,j}$ is the number of examples of class c_k covered by $C_{i,j}$). In addition, a default class (the majority class in the training data) is stored with T_{multi}. Then, given an (unseen) example e and a multi-class theory T_{multi}, we use the following algorithm to decide on the class of e:

- initialise the vector V of length m with each V_k=0
- for each conjunct $C_{i,j}$ in T_{multi}, if $C_{i,j}$ covers e, add $V_{i,j}$ to V.
- if no $C_{i,j}$ covers e (thus all V_k are 0), return the default class
  else, let V_k be the maximum in V, and return class k

5 Discretization

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

Procedures currently used to handle numbers in ILP and those used in older versions of CLAUDIEN[5], are quite expensive. The reason is that for each candidate clause, all values for a given numeric variable have to be generated and considered in tests. In large databases, the number of such values can be huge, resulting in a high branching factor of the search. Furthermore, discretization is done at runtime, i.e. it is repeated for every candidate clause. If one clause is a refinement of another one, a lot of redundant work may be done.

What we propose is to generate beforehand some interesting thresholds to test upon. Thresholds are thus computed only once (instead of once for each candidate clause considered). The number of interesting thresholds (to be considered when refining clauses) is also kept to a minimum, yielding a smaller branching factor. This has also yielded positive results in attribute value learning, cf. [2].

Though we present the procedure as applied in the ICL system, it also generalizes to other ILP systems The discretization procedure is tied with the DLAB parameter of ICL, which defines the syntax of the clauses that may be part of a hypothesis. In the template of section 2.2, the user has specified some possible thresholds for ICL. But where do they come from? Up to now, the user had to specify them. In most applications, this is not straightforward. We extend ICL with the capability to produce the possible thresholds itself.

When looking at the DLAB-templates it is often possible to identify a number of meaningful sub-clauses. We will call such sub-clauses *queries*. One could consider each such query that involves a numeric argument as a kind of numeric attribute. There is one important difference with regard to attribute value learning: one example may have multiple values for such a numeric query or attribute.

In our approach to discretization, the user has to identify the relevant queries and the variables for which the values are to be discretized. In DLAB:

```
dlab_template(
 ’false <-- 0-len:[len-len:[lumo(Lumo), lt(Lumo, c_lumo)],
                   len-len:[atom(A1, Elem1, Type1, Charge1),
                            lt(Charge1, c_charge) ]]’).
dlab_query(c_lumo, 1-1, discretize(lumo(Lumo), Lumo)).
dlab_query(c_charge, 1-1,
           discretize(atom(A1, Elem1, Type1, Charge1), Charge1)).
```

The resulting numeric attributes are then discretized using a simple modification of Fayyad and Irani's method, and the result is fed back into the DLAB template. The details of the Fayyad and Irani's method can be found in [11] and [9].

Fayyad and Irani's stopping criterion, which is based on the minimal description length principle, is very strict, in the sense that the method generates very few subintervals. When applying this criterion in ICL almost no subintervals would be generated. Therefore, we have chosen to let the method take as a parameter the desired number of thresholds to be generated (default 20). A second adaptation made to Fayyad and Irani's method specifically concerns nondeterminacy. Due to the fact that an example may have multiple or no values for a numeric attribute, we use a sum of weights instead of the number of examples in the appropriate places of Fayyad and Irani's formulae (i.e. when we count the real values that are less than a threshold, we sum their weights). 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.

6 Experiments

We have done experiments in three domains.

The data in the mutagenesis domain (see [17]) consists of 188 molecules, of which 125 are active (thus mutagenic) and 63 are inactive. A molecule is described by listing its atoms `atom(AtomID,Element,Type,Charge)` (the number of atoms differs between molecules, ranging from 15 to 35) and the bonds `bond(Atom1,Atom2,BondType)` between atoms. For the experiments, we have used the same four sets of background knowledge as in [17].

The data set for finite element mesh design [8], consists of 5 structures and has 13 classes (= 13 possible number of partitions for an edge in a structure). In total, the 5 structures consist of 278 edges (each edge is taken as an example). The background knowledge is relatively large and contains information on edge types, boundary conditions, loadings and the geometry of the structure.

The task in the biodegradability domain [10] is to predict the half-time of aqueous biodegradation of a compound from its chemical structure. The biodegradation time has been discretized into 4 classes: fast, moderate, slow and resistant. The structure of a compound is represented by facts about atoms and bonds, much like in the mutagenesis domain. An additional background predicate calculates the molecular weight (a real number) for each compound.

We have performed all the experiments with version 3 of ICL for Solaris2.5. The timings we give have been measured on a SUN Ultra 2 (168Mhz). The accuracies have been estimated using a n-fold cross-validation (mostly 10-fold). In some experiments, we used four different settings: for S_1 and S_2 the heuristic is set to laplace, for S_3 and S_4 to m_estimate (with parameter m set to 2). The significance level is set to 0.99 for S_1 and S_3 and to 0.90 for S_2 and S_4.

The mutagenesis domain requires the use of discretization, as it has real numbers. The mesh design domain requires the multi-class extension of ICL, as 13 classes are present. The biodegradability domain requires the use of both.

6.1 The Mutagenesis Experiments

Despite the fact that the multi-class feature of ICL is not needed for the mutagenesis domain, it turns out to be helpful. In table 1 we see that the multi-class theory always has a higher predictive accuracy than the CNF/DNF theory.

Muta	**Accuracies** (%)						**Timings** (s)			
	DNF	CNF	multi-DNF	Progol	Foil	TILDE	ICL	Progol	Foil	TILDE
BG1	79.3	69.7	81.4	76	61	75	276	117039	4950	93
BG2	80.3	72.3	81.9	81	61	79	423	64256	9138	355
BG3	85.6	82.4	86.2	83	83	85	440	41788	0.5	221
BG4	85.1	86.2	88.8	88	82	86	673	40570	0.5	651

Table 1. Accuracies and timings for the four different backgrounds of the mutagenesis data, with setting S_2 (the other three settings give similar results), using manual discretization. (The results for PROGOL, FOIL and TILDE have been taken from [1].)

The results in table 1 are obtained by supplying ICL with the possible boundaries for real-valued variables manually (the language looks like the dlab in section 2.2, but is more complex). Herefore, we needed some insight in the data. But what if we do not have this knowledge? We can then use the discretization feature of ICL. Table 2 shows the results of using the same language bias as in table 1, but using the discretization feature of ICL on the numerical variables. The results are comparable to the ones in table 1. We also tried to use all values appearing in the data (a very naive approach) for background 2. Learning one multi-theory then takes about 10 hours, which is about 30 times slower as compared to the other experiments! The accuracy is more or less the same.

Muta	**Accuracies (%)**			**Timings (s)**
with discretization	DNF	CNF	multi	multi
BG2	79.8	78.2	83.0	835
BG3	85.6	84.0	86.2	840
BG4	84.6	84.6	86.2	947

Table 2. Accuracies and timings of ICL for three different backgrounds of the mutagenesis data, with setting S_2 and using the **discretization** feature.

6.2 The Mesh Design Experiments

In this domain, the use of the multi-class extension is necessary as there are 13 classes. We performed tow experiments for each of the four settings, the results of which can be found in table 3:

- Exp_1: 5 runs have been done, each time using the edges of one structure for testing, and the edges from the other four structures for learning;
- Exp_2: a 10-fold cross-validation.

Mesh	**Accuracies (%)**				**Timings (s)**			
	S_1	S_2	S_3	S_4	S_1	S_2	S_3	S_4
Exp_1	46.8	49.3	50.0	49.3	206	211	315	332
Exp_2	65.1	64.7	66.5	70.1	192	190	223	307

Table 3. Results of ICL on the Mesh data (learning a multi-class).

When comparing the performance of ICL on Exp_1 with other learning systems (results taken from [1]), ICL has the highest predictive accuracy: ICL (50%), TILDE (36%), FOIL (21%), INDIGO (38%), FFOIL (44%) and FORS (31%).

6.3 The Biodegradability Experiments

While extensive experimentation has been conducted in the other two domains, the biodegradability domain is a relatively new one. It requires both the multi-class and the discretization facilities of ICL. Discretization is performed on the molecular weight and the charge values for individual atoms. The maximum number of thresholds is set to 20 (the current default).

The best result in this domain was achieved with setting S_4: an accuracy of 58.1% as measured by the leave-one-out procedure. These are the best results achieved so far in this domain (at the time of writing this paper). The problem is difficult because of the small number of very diverse examples.

7 Conclusion

We have indicated how several well-established attribute value learning techniques can be upgraded for use in the ILP system ICL. This includes the relation between CNF and DNF, discretization and multi-class problems.

We also showed with some experiments in 3 domains (mutagenesis, biodegradability and mesh) that ICL performs very good in these three domains.

In the future, we intend to do more experiments in other domains. Also, we might try other techniques for learning multi-classes, and look further into the possibilities of discretization.

We have proposed several extensions of the ICL ILP system, based on techniques developed for attribute-value approaches. These include the possibility to learn constraints on either CNF or DNF, handle multiple classes, and handle real numbers by using discretization.

These extensions are important for the use of ICL in practical domains, as demonstrated by our experimental evaluation of the extended ICL on three practical domains. Multi-class learning can help even when there are only two classes. Discretization increases efficiency at no cost to accuracy. Finally, learning DNF may sometimes be advantageous to learning CNF, due to the differences between learning the positive and learning the negative concept.

Regarding directions for further work, we note that multiple rules/theories are currently combined using the same approach as in CN2. It might be useful to look into different possibilities to merge several theories in one multi-class theory. We might look, for example, into the Bayesian approach used in [16].

Acknowledgements

Wim Van Laer and Luc De Raedt are supported by the Fund for Scientific Research, Flanders. Sašo Džeroski is supported by the Slovenian Ministry of Science and Technology. This research is also part of the ESPRIT project no. 20237 on Inductive Logic Programming II.

The Mutagenesis and Mesh datasets are made public by King and Srinivasan [17] resp. Dolšak [8], and are available at the ILP data repository [13].

More info on ICL: `http://www.cs.kuleuven.ac.be/~wimv/ICL/main.html`.

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.

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. P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In Yves Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 151–163. Springer-Verlag, 1991.
4. 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.
5. L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.
6. L. De Raedt and S. Džeroski. First order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.
7. 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.
8. 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.
9. 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.
10. S. Džeroski, B. Kompare, and W. Van Laer. Predicting biodegradability from chemical structure using ILP. Submitted.
11. 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.
12. M. Genesereth and N. Nilsson. *Logical foundations of artificial intelligence*. Morgan Kaufmann, 1987.
13. D. Kazakov, L. Popelinsky, and O. Stepankova. ILP datasets page [`http://www.gmd.de/ml-archive/datasets/ilp-res.html`] , 1996.
14. J.W. Lloyd. *Foundations of logic programming*. Springer-Verlag, 2nd edition, 1987.
15. R.J. Mooney. Encouraging experimental results on learning cnf. *Machine Learning*, 19:79–92, 1995.
16. U. Pompe and I. Kononenko. Probabilistic first-order classification, 1997. Submitted.
17. 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.
18. L. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

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