Inductive Constraint Logic *

Luc De Raedt and Wim Van Laer

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

Abstract. A novel approach to learning first order logic formulae from positive and negative examples is presented. Whereas present inductive logic programming systems employ examples as true and false ground facts (or clauses), we view examples as interpretations which are true or false for the target theory. This viewpoint allows to reconcile the inductive logic programming paradigm with classical attribute value learning in the sense that the latter is a special case of the former. Because of this property, we are able to adapt AQ and CN2 type algorithms in order to enable learning of full first order formulae. However, whereas classical learning techniques have concentrated on concept representations in disjunctive normal form, we will use a clausal representation, which corresponds to a conjuctive normal form where each conjunct forms a constraint on positive examples. This representation duality reverses also the role of positive and negative examples, both in the heuristics and in the algorithm. The resulting theory is incorporated in a system named ICL (Inductive Constraint Logic).

1 Introduction

Present work on inductive concept-learning is usually classified as belonging to either the attribute value learning paradigm (see [Quinlan, 1986; Michalski, 1983; Clark and Niblett, 1989], etc.) or the inductive logic programming paradigm [Muggleton and De Raedt, 1994; Muggleton, 1992]. The differences between the two paradigms are due to the representation formalism employed. Although the inductive logic programming paradigm is generally believed to be more expressive, there is no straightforward and generally accepted way to represent an attribute value learning problem as an inductive logic programming task. A possible explanation for this comes from the fact that in attribute value learning examples are true and false interpretations (models and non-models, or positive and negative examples) of a target theory, whereas in inductive logic programming, examples are true and false facts (or clauses). Recently, [De Raedt and Džeroski, 1994] have suggested this is as a possible explanation why PAC-learning results for inductive logic programming are so hard to obtain and mostly negative. They have also sketched a PAC-learning algorithm to learning (a certain class of) clausal theories from models (i.e. positive examples) of that theory. In this paper, we will further the ideas of De Raedt and Džeroski from a practical perspective. To this aim, we will define a notion of concept-learning in first order logic, in which examples are true and false interpretations of a target theory, and the target theory is a set of clauses. It will be convenient to view

* Generated on June 21, 1995. To appear in ALT-95

each clause in the target theory as a constraint, hence the term *inductive constraint logic*. Indeed, the aim of learning will be to induce a set of clauses (or constraints) such that every positive example p is true for *all* of the clauses (i.e. p is covered by all the clauses) and every negative example n is false for *at least one* of the clauses . This corresponds to learning a conjunctive normal form instead of a disjunctive normal form. Therefore the role of positive and negative examples is swapped, as compared to the usual covering approach of AQ and CN2 type systems. Nevertheless, this will enable us to develop the ICL algorithm, which is essentially a covering approach with the role of positive and negative examples reversed. Furthermore, as examples are interpretations, most of the other attribute value learning techniques, such as noise handling heuristics, will nicely upgrade towards our framework. Some of these ideas (like the swapping of positives and negatives) can also be found in [Mooney, 1995]. However, our approach is more general, as we work with a first order logic representation and with examples as interpretations.

The paper is organized as follows: in Section 2, we introduce the inductive constraint logic framework; in Section 3, we sketch the ICL algorithm for inducing clausal theories, in Section 4, we report on some preliminary experiments, and in Section 5 we discuss related and further work and finally conclude in Section 6.

2 Inductive constraint logic: definitions and framework

We assume familiarity with first order logic and model theory (see [Lloyd, 1987; Genesereth and Nilsson, 1987] for an introduction).

A first order alphabet is a set of predicate symbols, constant symbols and functor symbols. A clause is a formula of the form $A_1; ...; A_m \leftarrow B_1, ..., B_n$ where the A_i and B_i are logical atoms. 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 $f(t_1, ..., t_k)$ immediately followed by a bracketed n-tuple of terms t_i. Constants are function symbols of arity 0. *Functor-free* clauses are clauses that contain only variables as terms.

The above clause can be read as A_1 or ... or A_m if B_1 and ... and B_n. All variables in clauses are universally quantified, although this is not explicitly written. Extending the usual convention for *definite clauses* (where $m = 1$), we call $A_1, ..., A_m$ the *head* of the clause and $B_1, ..., B_n$ the *body* of the clause. A *fact* is a definite clause with an empty body, ($m = 1,\ n = 0$). Throughout the paper, we shall assume that all clauses are *range restricted*, which means that all variables occurring in the head of a clause also occur in its body.

A *Herbrand interpretation* over a first order alphabet is a set of ground facts constructed with the predicate, constant and functor symbols in the alphabet. A Herbrand interpretation I is a model for a clause c if and only if for all grounding substitutions θ of c : $body(c)\theta \subset I \rightarrow head(c)\theta \cap I \neq \emptyset$. We also say c is true in I or c makes the interpretation I true, or even I is a true interpretation for c. If a herbrand interpretation I is not a model for a clause c, we say that c is false in interpretation I. A Herbrand interpretation I is a model for a clausal theory T if and only if it is a model for all clauses in T. Roughly speaking, the truth of a clause c in an interpretation I can be determined by running the query $?-body(c), not\ head(c)$

on a database containing I using a theorem prover (such as Prolog). If the query succeeds, the clause is false in I. If it finitely fails, the clause is true.

By now we can intuitively describe our inductive constraint logic setting. One starts from a set of positive and negative examples (true and false interpretations or models and non-models of a target theory) and aims at finding a clausal theory for which all positive examples are models and none of the negative examples is a model. A definite clause background theory can be used as follows. Rather than starting from complete interpretations of the target theory, examples are a kind of partial interpretations I (set of facts) and are completed by taking the minimal Herbrand model $M(B \cup I)$ of background theory B plus the partial interpretation I. Note that the completion need not be computed explicitly, but can be left to the theorem-prover[2]. The minimal Herbrand model of a definite clause theory contains the set of all ground facts that are logically entailed by that theory[3]. Formally, this setting thus becomes:

- **Given**
  - P is a set of interpretations such that for all $p \in P$, $M(B \cup p)$ is a true interpretation of the unknown target theory;
  - N is a set of interpretations such that for all $n \in N$, $M(B \cup n)$ is a false interpretation of the unknown target theory;
  - a definite clause background theory B;
- **Find**: a clausal theory H such that
  - for all $p \in P$, $M(B \cup p)$ is a true interpretation of H (Completeness)
  - for all $n \in N$, $M(B \cup n)$ is a false interpretation of H (Consistency).

In the following, each example will be completed implicitly with the background knowledge. So whenever we say an example e is true (false) for a clause (or theory), we mean that $M(B \cup e)$ is true (false) for that clause (theory).

This setting is illustrated in Examples 1, 2 and 3.

Example 1. The well-known autolander problem (from the Irvine database) is described by a table (see figure 1) in attribute value representation (only a part is shown).

This attribute value learning problem can directly be specified in terms of our framework. E.g. the interpretation I_1= {stability(stab), error(mm), sign(pp), wind(tail), magnitude(strong), visibility(yes)} (last line) corresponds to a positive example, and I_2 = {stability(stab), error(mm), sign(pp), wind(head), magnitude(strong), visibility(yes)} is a negative example. Given this problem (with no background knowledge), the ICL technique derives the following clausal theory for the positive class.

visibility(no) $\leftarrow$ *stability(xstab)*
visibility(no) $\leftarrow$ *magnitude(out-of-range)*

[2] Using Prolog, one could assert background theory and interpretation into the knowledge base, and run the query $? - \mathit{body}(c)$, *not* $\mathit{head}(c)$, in order to test whether a clause c makes an interpretation true or not.

[3] The setting can easily be extended to handle program clauses including negation, provided that the model of background theory and interpretation is unique. Such a semantics is provided for instance by the preferred model semantics.

stability	error	sign	wind	magnitude	visibility	CLASS
xstab	XL	nn	head	Low	no	$\oplus$
...	...	...	...	...	no	$\oplus$
xstab	XL	nn	head	Low	yes	$\ominus$
xstab	...	...	...	...	...	$\ominus$
stab	LX	nn	head	Low	yes	$\ominus$
stab	LX	...	...	...	yes	$\ominus$
stab	XL	nn	head	Low	yes	$\ominus$
stab	XL	...	...	...	yes	$\ominus$
stab	MM	nn	tail	Low	yes	$\ominus$
stab	MM	nn	tail	...	yes	$\ominus$
xstab	XL	nn	head	OutOfRange	yes	$\ominus$
...	...	...	...	OutOfRange	yes	$\ominus$
stab	SS	nn	head	Low	yes	$\oplus$
stab	SS	...	...	Low	yes	$\oplus$
stab	SS	nn	head	Medium	yes	$\oplus$
stab	SS	...	...	Medium	yes	$\oplus$
stab	SS	nn	head	Strong	yes	$\oplus$
stab	SS	...	...	Strong	yes	$\oplus$
stab	MM	pp	head	Low	yes	$\oplus$
stab	MM	pp	head	Medium	yes	$\oplus$
stab	MM	pp	tail	Low	yes	$\oplus$
stab	MM	pp	tail	Medium	yes	$\oplus$
stab	MM	pp	head	Strong	yes	$\ominus$
stab	MM	pp	tail	Strong	yes	$\oplus$

Fig. 1. The Autolander input

visibility(no) $\leftarrow$ *error(xl)*
visibility(no) $\leftarrow$ *error(lx)*
visibility(no);error(ss) $\leftarrow$ *sign(nn)*
visibility(no);error(ss);wind(tail) $\leftarrow$ *magnitude(strong)*

Notice that I_1 is true for all clauses, whereas I_2 is false for the last clause and thus for the clausal theory.

Example 2. The *Bongard Problems* are general problems developed by the Russian scientist M. Bongard in his book *Pattern recognition* (a nice overview can be found in Gödel, Escher and Bach, Douglas R. Hofstadter). Each problem exists of 12 figures, six of class $\oplus$ and six of class $\ominus$. One example problem can be found in figure 2, based on the *Bongard Problem 47*. The goal is to discriminate between the two classes.

Each figure consists of lines, points, squares, triangles, circles,... which can be white, black, small, large, horizontal,... So each of these figures can be described by a set of facts. Take for instance the upper left figure in our example problem. It consists of a small triangle, pointing up, which is in a large circle. This figure can thus be described as : I = {*triangle(f1), small(f1), up(f1), circle(f2), large(f2), in(f1,f2)*}.

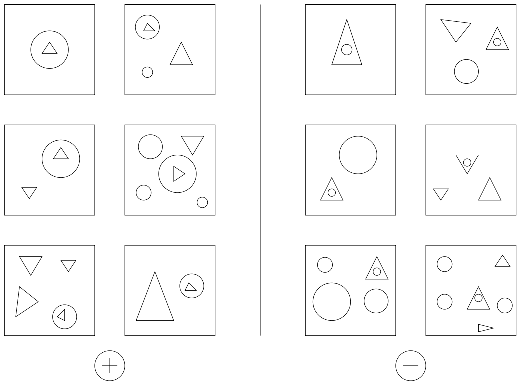

Fig. 2. A Bongard Problem

In our setting, each interpretation (example) is a set of facts, describing one figure. The following theory is consistent and complete (found by ICL in a few seconds) :

$false \leftarrow circle(Y), in(Y,X)$

This constraint says that for each figure of class $\oplus$, there is no circle that is inside another object. This can also be read as : if a figure has a circle inside another object, it is of class $\ominus$.

Example 3. A well known problem in the context of inductive logic programming is the card game of Eleusis. The problem was first decribed by Dietterich and Michalski [Dietterich and Michalski, 1985], and later employed by Quinlan [Quinlan, 1990], and Lavrac and Dzeroski [Lavrač and Džeroski, 1994]. There are two players. One of them has to guess the secret rule the other player has in mind.

main line	J♣	4♣	Q♡	3♠	Q♢	9♡	Q♣	7♡	Q♢	9♢	Q♣	3♡	K♡
side lines	K♣	5♠				4♠		10♢					
		7♠											
main (ctd)	4♣	K♢	6♣	J♢	8♢	J♡	7♣	J♢	7♡	J♡	6♡	K♢	
mainline	4♡	5♢	8♣	J♠	2♣	5♠	A♣	5♠	10♡				
side line	7♣	6♠	K♣	A♡		6♣		A♠					
	J♡	7♡	3♡	K♢									
	4♣	2♣		Q♠									
	10♠	7♠											
	8♡	6♢											
	A♢	6♡											
	2♢	4♣											

Fig. 3. Two Eleusis layouts

Two example sequences can be found in figure 3 (from [Lavrač and Džeroski, 1994]). Two consecutive cards on the main line satifies the secret rule. These can be used as positive examples. Each card on a side line is an illegal successor of the corresponding card on the main line. So these two cards can be seen as a negative example. Each example (a set of two cards) is translated (as in [Van Laer *et al.*, 1994]) into one fact of the form: $canfollow(R_2, S_2, R_1, S_1)$, which states that a card of rank R_2 and suit S_2 can follow a sequence ending with a card of rank R_1 and suit S_1. In this way, each interpretation contains one fact. For example in the first sequence, one positive model might be $canfollow(Q, \heartsuit, 4, \clubsuit)$, while a negative model might be $canfollow(5, \spadesuit, 4, \clubsuit)$.
Note that these interpretations are (implicitly) completed with the background knowledge, which contains the definitions of *red, black, samecolor, number, face, precedesrank, lowerrank, precedessuit.*

In the first sequence, the intended rule was "Play alternate face and non-face cards". Following our setting, ICL found the following constraints on the positive examples (in about 10 sec):

$face(R_2) \leftarrow canfollow(R_2,S_2,R_1,S_1), number(R_1)$
$number(R_2) \leftarrow canfollow(R_2,S_2,R_1,S_1), face(R_1)$

which states the intended rule. Contrast this with the kind of rules LINUS or FOIL learns:

$canfollow(R_2,S_2,R_1,S_1) \leftarrow face(R_2),\ number(R_1)$
$canfollow(R_2,S_2,R_1,S_1) \leftarrow face(R_1),\ number(R_2)$

The intended rule for the second layout is "Play a higher card in the suit preceding that of the last card; or, play a lower card in the suit following that of the last card". ICL learns three constraints on the positive models:

precedes_suit(S1,S2) ← can_follow(R2,S2,R1,S1),lower_rank(R1,R2)
lower_rank(R1,R2) ← can_follow(R2,S2,R1,S1),red(S1)
precedes_suit(S2,S1) ← can_follow(R2,S2,R1,S1),lower_rank(R2,R1)

A nicer result is obtained when the positive and negative models are switched, which gives us constraints on the negative models :

lower_rank(R2,R1) ← can_follow(R2,S2,R1,S1),precedes_suit(S1,S2)
lower_rank(R1,R2) ← can_follow(R2,S2,R1,S1),precedes_suit(S2,S1)

This is very simular to the result obtained by LINUS.

The view that examples are interpretations and that the aim is to discriminate between two classes of examples is similar to classical learning from positive and negative examples. This is illustrated by the fact that an attribute value learning task (the Autolander problem) can be represented in a straightforward way in our framework. Furthermore, for many complex and structural induction problems the view that examples are interpretations is a very natural one, cf. for instance the drug activity problem [King *et al.*, 1992] the mutagenicity [Srinivasan *et al.*, 1994] and Michalski's classical eastbound - westbound train problem [Michalski and Stepp, 1983] that was recently used as the basis for a machine learning competition.

3 Inductive Constraint Logic: a first algorithm

The key observation to arrive at the ICL algorithm is the following. Each clause belonging to the target theory will have to make *all* positive examples true, and will have to exclude *some* of the negatives. This contrasts with the classical attribute value learning setting where each conjunctive rule has to cover some of the positives and none of the negatives. This contrast is actually a duality because clauses are to be considered as disjunctions of literals, rather than as conjunctions. So, clausal theories correspond to a conjunctive normal form (a conjunction of disjunctions) whereas classical attribute value learning has concentrated on learning a disjunctive normal form (a disjunction of conjunctions). A conjunctive normal form formula is of roughly the following structure:

$$(l_{1,1} \vee \ldots \vee l_{1,n_1}) \wedge \ldots \wedge (l_{m,1} \vee \ldots \vee l_{m,n_m})$$

and for a disjunctive normal form formula:

$$(l_{1,1} \wedge \ldots \wedge l_{1,n_1}) \vee \ldots \vee (l_{m,1} \wedge \ldots \wedge l_{m,n_m}).$$

This duality is further illustrated in Example 4 for the autolander problem.

Example 4. The results generated by CN2 for the Autolander problem (negative class) is:

$(stability(xstab) \wedge visibility(yes)) \vee$
$(error(xl) \wedge visibility(yes)) \vee$
$(error(lx) \wedge visibility(yes)) \vee$
$(magnitude(outofrange) \wedge visibility(yes)) \vee$
$(error(mm) \wedge sign(nn) \wedge visibility(yes)) \vee$
$(error(mm) \wedge wind(head) \wedge magnitude(strong) \wedge visibility(yes))$

It is easily verified that the negation of this description corresponds to the clausal form formula in Example 1. The negation of the negative class in DNF will for boolean learning problems always correspond to the positive class in CNF. However, when variables appear in clauses, complications arise because of quantifiers. As a result, the negation of the negative class may no longer be expressible in clausal form, cf. also Section 5.

The duality between CNF and DNF can easily be exploited by inversing well-known algorithms (such as AQ and CN2) that learn disjunctive normal forms to learn CNF formula (cf. also [Haussler, 1988]). It is because of this property that we will pursue a covering approach on the negatives instead of the positives, and also generally invert the role of positives and negatives. Also, in the heuristics the role of positives and negatives will be swapped. Furthermore, as examples are interpretations, the heuristics will count interpretations. This contrasts ICL from other ILP algorithms (such as FOIL [Quinlan, 1990]) where one counts facts and/or substitutions. The ICL algorithm derived from CN2 [Clark and Niblett, 1989] by exploiting this duality is shown in Figures 4 and 5

Figure 4 contains the covering algorithm. Given a background theory B and a set of positive examples P and negative examples N, the algorithm produces a theory H. Initially the hypothesis is empty, and the algorithm repeatedly tries to find interesting clauses c to add to the hypothesis. Each clause added will make all positive examples true and will make some negatives false. After a clause is found, the negatives that are false interpretations for the clause (i.e. not covered by the clause) are discarded. This process continues until no significant clause can be found, or a complete and consistent hypothesis is discovered.

Inductive-Constraint-Logic(P,N,B);

1. Initialize $H := \emptyset$
2. **repeat until** *best clause c* not found **or** N is empty
   (a) find *best clause c*
   (b) **if** *best clause c* found **then**
      i. add c to H
      ii. remove from N all interpretations that are false for c
3. **return** H

Fig. 4. ICL covering algorithm

Find-Best-Clause(P,N,B);

1. Initialize $Beam := \{\mathit{false} \leftarrow \mathit{true}\}$
2. Initialize $BestClause := \emptyset$
3. **while** $Beam$ is not empty **do**
   (a) Initialize $NewBeam := \emptyset$
   (b) **for** each clause c in $Beam$ **do**
      i. **for** each refinement Ref of c **do**
         A. **if** Ref is better than $BestClause$
            and Ref is statistically significant
            then $BestClause := Ref$
         B. **if** Ref is not to be pruned
            then
            – add Ref to $NewBeam$
            – **if** size of $NewBeam > MaxBeamSize$
              then remove worst clause from $NewBeam$
   (c) $Beam := NewBeam$
4. **return** $BestClause$

Fig. 5. ICL beam search algorithm

The search algorithm for the best clause is shown in figure 5. As in CN2, we use beam search as search strategy. A classical refinement operator under θ-subsumption (Plokin 70, Shapiro 83) is used together with CLAUDIEN's mechanism to specify the declarative bias (i.e. the syntax of well-formed clauses in hypotheses), cf. [Van Laer *et al.*, 1994; Adé *et al.*, 1995] and below. (Notice that whenever a clause c is true in an interpretation I, all refinements of c are also true in I.)

The search starts with the most general clause of the refinement graph (namely $false \leftarrow true$, for wich each example model is false). During the search, ICL keeps a *Beam* of candidate clauses and the best clause found sofar (*BestClause* in 2.). At each step in the beam search, all the possible refinements Ref of the clauses in *Beam* are considered and evaluated (see A and B in figure 5). Depending on the evaluation, Ref becomes the *CurrentBest* and/or Ref is added to *NewBeam* (such that only the *MaxBeamSize* best candidate clauses remain in the beam).

ICL makes use of two heuristic evaluation functions. The first one estimates the *quality of a clause*. This measure is used to direct the search (as only the best *MaxBeamSize* clauses are placed in the beam) and to determine the best clause found during the beam search. The quality of a clause c is determined by the probability that an example interpretation is negative, given that clause c is false in the interpretation, i.e. $p(\ominus|\overline{c})$. Notice the difference with classical attribute value learning, where $p(\oplus|c)$ is used (see [Lavrač and Džeroski, 1994]). We use the Laplace estimate (as default) to measure this probability :

$$HV(c) = p(\ominus|\overline{c}) = \frac{n^{\ominus}(\overline{c}) + 1}{n(\overline{c}) + 2}$$

where $n(\overline{c})$ is the number of examples that are false for clause c and $n^{\ominus}(\overline{c})$ is the number of negative examples that are false for c.

The second evaluation function tests whether a clause is statistically significant, to ensure that the clause represents a genuine regularity in the examples and not a regularity which is due to chance (see [Lavrač and Džeroski, 1994; Clark and Niblett, 1989]). The statistical significance test is based on the likelihood ratio statistic, which is also employed in CN2. This gives

$$LR(c) = 2n(c) \;\times\; (p(\oplus|c)log(\frac{p(\oplus|c)}{p_a(\oplus)}) + p(\ominus|c)log(\frac{p(\ominus|c)}{p_a(\ominus)}))$$

where $n(c)$ is the number of interpretations covered by c (true for c), $p_a(\oplus)$ and $p_a(\ominus)$ are the prior probabilities of classes $\oplus$ and $\ominus$ (= relative frequency of positive and negative examples in the training set), $p(\oplus|c)$ is the probability (estimated by relative frequency) that an example is positive given that c is true for that example, and $n(c)$ is the total number of examples satisfying clause c. This statistic is distributed approximately as χ^2 with one degree of freedom. We say that a clause c is significant, if $LR(c) > T$, with T a certain significance treshold (for example 6.64 for 99 per cent significance).

Based on these two heuristics, we can do some additional pruning. First of all, a clause c can be pruned if no refinements of c can become better than the best clause at that moment (the best value we can achieve with further refinements of a clause, is a clause that is false for the same negative models and true for all positive models) (cfr. gain pruning in FOIL). Secondly, we can stop refining a clause when it's not possibly statistically significant (cfr. mFoil [Lavrač and Džeroski, 1994]).

Notice that the refinement operator in ICL is a classical one based on Plotkin's θ-subsumption [Plotkin, 1970]. Though a complete refinement operator for clausal logic could be employed by ICL (as is given in for instance in [Shapiro, 1983; van der Laag and Nienhuys-Cheng, 1993], we choose to enhance ICL with a declarative bias mechanism for restricting the search space. The declarative bias is the same as that in the CLAUDIEN system [Van Laer *et al.*, 1994] and in NINA [Adé *et al.*, 1995]. Basically, clause models are used to define the syntax of clauses that can appear in hypotheses. From the models, one can automatically derive a refinement operator that only generates clauses that are allowed by the syntax. A full discussion of this declarative bias mechanism is outside the scope of this paper, but see [Adé *et al.*, 1995; Van Laer *et al.*, 1994] for more details.

4 Experiments

In this section, we report on an experiment in the King-Rook-King domain of [Muggleton *et al.*, 1989]. The experiment mainly serves to demonstrate the performance of ICL on a larger domain involving noise. The same experimental setting and the same data as [Lavrač and Džeroski, 1994] was employed. The significance level of ICL was set to 99 per cent. There were for all different noise levels, 5 runs of 100 examples (60 positive examples and 40 negatives), whereas the testset contained 5000 examples. The results are averaged over the 5 runs and compared to the results for mFOIL using the Laplace heuristic (see Figure 6 and 7)[4]. A noise level of x per cent in the arguments means that in x per cent of the examples the values of one of the arguments was replaced by a random value. Similarly, a noise level of x per cent in the class means that in x per cent of the examples the classification was assigned randomly.

In order to make the comparison easier, we learned the negative concept. As outlined in the example for the Autolander problem, the negation of the negative concept in the form of clausal constraints (CNF) corresponds roughly to the definition of the positive concept in DNF format. An example output (for the noise free runs) is:

false ← *wk(WKf, WKr), wr(WRf, WRr), bk(BKf, BKr), WRf=BKf*
false ← *wk(WKf, WKr), wr(WRf, WRr), bk(BKf, BKr), WRr=BKr*
false ← *wk(WKf, WKr), wr(WRf, WRr), bk(BKf, BKr), adj_file(WKf, BKf), adj_rank(WKr, BKr)*
false ← *wk(WKf, WKr), wr(WRf, WRr), bk(BKf, BKr), WKf=BKf, adj_rank(WKr, BKr)*
false ← *wk(WKf, WKr), wr(WRf, WRr), bk(BKf, BKr), adj_file(WKf, BKf), WKr=BKr*
false ← *wk(WKf, WKr), wr(WRf, WRr), bk(BKf, BKr), WKf=BKf, WKr=BKr*
false ← *wk(WKf, WKr), wr(WRf, WRr), bk(BKf, BKr), adj_rank(WRr, BKr), WKf=WRf*

As one can see ICL performs roughly the same as mFOIL on these data.

[4] For ease of comparison we choose to compare with the Laplace estimate in mFOIL and not with the best m-value, because ICL also uses the Laplace heuristic though ICL could also employ the m-estimate.

System	Noise in Arguments							
	0	5	10	15	20	30	50	80
mFOIL - Laplace	95.14	90.11	80.41	78.32	73.66	72.57	67.91	59.20
ICL	95.26	90.78	79.04	78.46	72.22	72.36	66.44	62.14
System	Noise in Class							
	0	5	10	15	20	30	50	80
mFOIL - Laplace	95.14	92.71	88.90	86.40	84.58	83.75	73.74	60.25
ICL	95.26	92.78	88.84	86.12	82.28	83.04	76.84	63.96
System	Noise in Class and Arguments							
	0	5	10	15	20	30	50	80
mFOIL - Laplace	95.14	85.36	75.76	75.08	71.69	64.22	61.38	57.58
ICL	95.26	90.66	78.58	74.86	74.10	65.90	64.64	59.14

Fig. 6. Results of the King-Rook-King experiment

5 Discussion

In this section we discuss the strenghts and the weaknesses of ICL as compared to other approaches.

First, notice that ICL is meant to learn binary concepts from examples. As such it addresses the same task as propositional learners (but in a more expressive framework) and it does not learn logic programs from examples as many ILP systems do cf. [Muggleton and De Raedt, 1994]. In ILP, examples are usually true and false ground facts of a target predicate, and the result is a logic program. For instance when applying ILP to program synthesis, the examples show the input-output behaviour of the target predicate, and the resulting program can be used to produce the output for a given input. Inducing a logic program involves learning a more complex behaviour than inducing a classifier as the latter can only be used to produce a true/false answer.

A second difference with the majority of ILP systems, is that there is a clear view of how to use examples in heuristics. This is because in ICL one interpretation counts as one example. In many ILP systems (such as e.g. Quinlan's FOIL) it is unclear what counts as an example. At some points, FOIL counts facts, and at other points it counts substitutions. We believe this is one of the reasons that heuristics for attribute value learning are much better understood than for ILP. Our framework resolves this confusion and allows to carry over the results of a lot of research on using heuristics in attribute value learning to inductive constraint logic.

Third, let us mention that ICL assumes that the examples are finite sets of facts because it is decidable for all clauses and for all finite interpretations whether or not the interpretation is a model for the clause. Notice that this property also holds for recursive clauses. F.i. it is decidable whether or not $\mathrm{ancestor}(X,Y) \leftarrow \mathrm{parent}(X,Z) \wedge \mathrm{ancestor}(Z,Y)$ makes any finite interpretation true. This means that ICL has no problems learning recursive clauses, which contrasts again with most of the other ILP approaches. However, when interpretations may be infinite (f.i. in the presence of functor symbols), testing for coverage is no longer decidable. A (rather technical) approach to circumventing this problem based on [Rouveirol, 1994], is given in [De Raedt and Džeroski, 1994]. Another point of difference is that in ICL

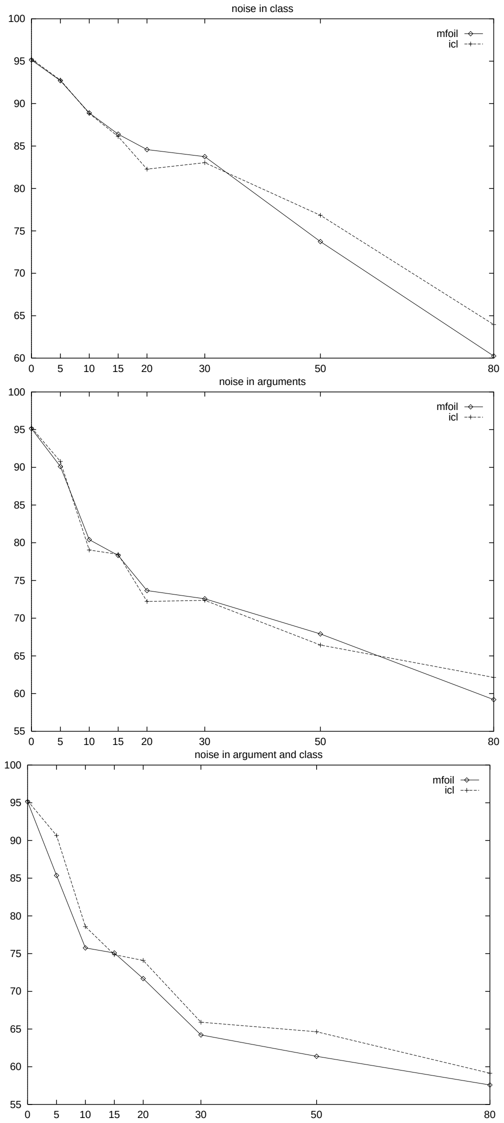

Fig. 7. Results of the King-Rook-King experiment

all clauses are considered independent constraints, whereas in ILP clauses may be mutually dependent, complicating coverage tests [De Raedt *et al.*, 1993].

Fourth, the clausal logic we employ allows to easily express some concepts which are hard (or impossible) to express using the normal inductive logic programming paradigm and definite clauses. For instance, a concept such as all wagons are black or white, can very naturally be represented by $black(X) \vee white(X) \leftarrow wagon(X)$.

Fifth, whereas some forms of existential quantification are easily handled in the normal inductive logic programmign setting using definite clauses, ICL cannot (yet) learn concepts involving existential quantifiers. A possible target concept could be the following expression in prenex conjunctive normal form

$$\exists W : wagon(W) \wedge black(W) \wedge small(W)$$

(In words: there is a wagon that is black and small. This concept could be expressed in normal ILP by a clause such as $positive(T) \leftarrow wagon(T, W) \wedge black(W) \wedge small(W)$.) The reason that this cannot directly be translated in clausal logic is that the translation from predicate calculus to clausal logic involves skolemization (see Genesereth and Nillson 87), which essentially introduces names for all existential quantifiers. Thus the corresponding clausal logic formula (after skolemization) would be:

$$wagon(sk) \wedge small(sk) \wedge wagon(sk)$$

Now it is easy to find interpretations that are models for the prenex conjunctive normal form but not for skolemized version of it. At present we are working on an extension of ICL that learns formula in prenex conjunctive normal form rather than clausal form. One complication in this context is that disjunctions (or clauses) can no longer be considered independently of each other. F.i. in the above target concept, the three constraints (wagon, black and small) have to be considered at the same time in order to test coverage.

6 Related Work and Conclusions

ICL forms the practical counterpart to the PAC-learning framework presented by De Raedt and Džeroski who also view examples as interpretations of a clausal theory. Secondly, some of the ideas on reversing the role of positives and negatives when going from CNF to DNF or vice versa, were presented in [Haussler, 1988] and in [Mooney, 1995]. Given Haussler's results it was relatively easy to turn ideas of Clark and Niblett's CN2 into ICL. Finally, we believe ICL is also similar in spirit as Michalski's work [Michalski, 1983] on learning structural concept definitions, in that it addresses the same learning task. However, the use of a formal logical framework as well as more advanced attribute value learning techniques distinguishes ICL from AQ.

We have outlined a novel approach to learn classification rules in the predicate calculus. Whereas the large majority of results of induction in first order logic views examples as ground facts or clauses, we view examples as interpretations. This corresponds to learning a conjunctive normal form instead of a disjunctive one. The presented technique seems promising for learning classification rules because it inherits many well-known techniques from attribute value learning as well as the much more expressive framework of inductive logic programming.

Just before submitting this article, it was brought to our attention that there are some unpublished results on learning propositional CNF formulas by Ray Mooney, which would be to appear in the MLJournal. In a final version of this paper, we hope to elaborate on this.

Acknowledgements

Luc De Raedt and Wim Van Laer are supported by the Belgian National Fund for Scientific Research and by the ESPRIT project no. 6020 on Inductive Logic Programming, co-financed by the Flemish Government under contract no. 93/14. They would like to thank Sašo Džeroski, Nada Lavrač and Maurice Bruynooghe for interesting discussions and earlier work that finally lead to this paper and Sašo Džeroski for supplying the data for the KRK experiment.

References

[Adé *et al.*, 1995] H. Adé, L. De Raedt, and M. Bruynooghe. Declarative Bias for Specific-To-General ILP Systems. *Machine Learning*, 1995. To appear.

[Clark and Niblett, 1989] P. Clark and T. Niblett. The CN2 algorithm. *Machine Learning*, 3(4):261–284, 1989.

[De Raedt and Džeroski, 1994] L. De Raedt and S. Džeroski. First order jk-clausal theories are pac-learnable. *Artificial Intelligence*, 70:375–392, 1994.

[De Raedt *et al.*, 1993] L. De Raedt, N. Lavrač, and S. Džeroski. Multiple predicate learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1037–1042. Morgan Kaufmann, 1993.

[Dietterich and Michalski, 1985] T.G. Dietterich and R.S. Michalski. Discovering patterns in sequences of events. *Artificial Intelligence*, 25:257–294, 1985.

[Genesereth and Nilsson, 1987] M. Genesereth and N. Nilsson. *Logical foundations of artificial intelligence*. Morgan Kaufmann, 1987.

[Haussler, 1988] D. Haussler. Quantifying inductive bias : AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177 – 221, 1988.

[King *et al.*, 1992] R.D. King, S. Muggleton, R.A. Lewis, and M.J.E. Sternberg. Drug design by machine learning: the use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Sciences*, 89(23), 1992.

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

[Lloyd, 1987] J.W. Lloyd. *Foundations of logic programming*. Springer-Verlag, 2nd edition, 1987.

[Michalski and Stepp, 1983] R.S. Michalski and R.E. Stepp. Learning from observation: conceptual clustering. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: an artificial intelligence approach*, volume 1. Tioga Publishing Company, 1983.

[Michalski, 1983] R.S. Michalski. A theory and methodology of inductive learning. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: an artificial intelligence approach*, volume 1. Morgan Kaufmann, 1983.

[Mooney, 1995] R.J. Mooney. Encouraging experimental results on learning cnf. *Machine Learning*, 19:79–92, 1995.

[Muggleton and De Raedt, 1994] S. Muggleton and L. De Raedt. Inductive logic programming : Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

[Muggleton *et al.*, 1989] S. Muggleton, M. Bain, J. Hayes-Michie, and D. Michie. An experimental comparison of human and machine learning formalisms. In *Proceedings of the 6th International Workshop on Machine Learning*, pages 113–118. Morgan Kaufmann, 1989.

[Muggleton, 1992] S. Muggleton, editor. *Inductive Logic Programming.* Academic Press, 1992.

[Plotkin, 1970] G. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

[Quinlan, 1986] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Quinlan, 1990] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[Rouveirol, 1994] C. Rouveirol. Flattening and saturation: Two representation changes for generalization. *Machine Learning*, 14:219–232, 1994.

[Shapiro, 1983] E.Y. Shapiro. *Algorithmic Program Debugging.* The MIT Press, 1983.

[Srinivasan *et al.*, 1994] A. Srinivasan, S.H. Muggleton, R.D. King, and M.J.E. Sternberg. Mutagenesis: Ilp experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[van der Laag and Nienhuys-Cheng, 1993] P.R.J. van der Laag and S.-H. Nienhuys-Cheng. Subsumption and refinement in model inference. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 95–114. Springer-Verlag, 1993.

[Van Laer *et al.*, 1994] W. Van Laer, L. Dehaspe, and L. De Raedt. Applications of a logical discovery engine. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, pages 263–274, 1994.

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