The role of feature construction in inductive rule learning

Peter A. Flach[1] and Nada Lavrač[2]

[1] Department of Computer Science
University of Bristol, United Kingdom
[2] Department of Intelligent Systems
Jožef Stefan Institute, Ljubljana, Slovenia

Abstract. This paper proposes a unifying framework for inductive rule learning algorithms. We suggest that the problem of constructing an appropriate inductive hypothesis (set of rules) can be broken down in the following subtasks: rule construction, body construction, and feature construction. Each of these subtasks may have its own declarative bias, search strategies, and heuristics. In particular, we argue that feature construction is a crucial notion in explaining the relations between attribute-value rule learning and inductive logic programming (ILP). We demonstrate this by a general method for transforming ILP problems to attribute-value form, which overcomes some of the traditional limitations of propositionalisation approaches.

1 Introduction

Rule learning tasks are typically approached as search problems, and hence the construction of candidate hypotheses is a crucial task in inductive rule learning algorithms. Traditionally, hypotheses are constructed by constructing rules, rules are constructed by constructing heads and bodies, and bodies are constructed by putting literals together. In this paper we suggest that it is more natural to consider rule bodies as consisting of *features*, which need to be constructed in a separate step. Feature construction is not usually distinguished in inductive rule learning (at least not as a *necessary* step), and we argue that this may result in unnecessary complications and loss of expressiveness, especially when rules are first-order clauses. Distinguishing feature construction as a separate subtask also leads to a new transformation-based learning approach by which many non-determinate logic programming problems can be transformed to attribute-value learning problems without loss of generality.

The paper is organised as follows. In Section 2 we briefly discuss the traditional trichotomy of hypothesis construction, rule construction, and body construction. Section 3 defines feature construction, concentrating on first-order features. Section 4 shows how this leads to a novel propositionalisation approach that can solve non-determinate relational learning tasks. Section 5 concludes.

2 Hypothesis construction, rule construction, and body construction

We start by considering the more or less traditional distinction between hypothesis construction, rule construction, and body construction. Although not every inductive rule learner distinguishes between these tasks in the same way, the distinction is conceptually justified because in each of these tasks different biases, search strategies, and heuristics may be applied.

As a hypothesis is a set of rules, the search space for hypotheses is 2^R where R is the search space for rules. In practice, however, an n-rule hypothesis is constructed by n separate rule constructions. For instance, the well-known covering algorithm constructs one rule at a time, removes the examples that are covered by that rule, and then iterates on the remaining examples. Notice that this is problematic in case the hypothesis language includes recursive first-order rules: in that case, either the example set has to be complete, or the rules have to be learned in a certain order (e.g., the base case first). In this paper we restrict attention to non-recursive stratified rule sets (i.e. there is no recursion through negation either), and we can safely assume that rule sets are learned by calling a rule learner several times.

The distinction between hypothesis construction and rule construction is important from the viewpoint of heuristics. Suppose we are addressing a two-class learning task, and we are using predictive accuracy to evaluate hypotheses, i.e. the sum of the relative frequencies of true positives and true negatives. Rule accuracy however is usually measured differently: the rule accuracy measure punishes a rule for being overly general but not for being overly specific, since an uncovered positive example can be taken care of by another rule. So rule accuracy is defined as the relative frequency of true positives among the instances covered by the rule.

Next, we consider the task of rule construction. We assume that rules are implications consisting of a body (antecedent) and a head (consequent). A typical approach to rule construction is to select a head corresponding to a class, and then constructing an appropriate body. Not all rule learning algorithms follow this approach: for instance, CN2 [2] maintains a beam of best bodies, evaluated w.r.t. information gain (i.e. with maximal improvement in the distribution of classes among the covered examples), and then assigning a rule head (class assignment) maximising classification accuracy. As another example, association rule learning algorithms, e.g. APRIORI [1], first search for the most promising bodies (frequent itemsets), and then construct association rules from two frequent itemsets such that one includes the other. In these two cases, body construction is the main step of rule construction.

Usually, a rule body is a conjunction of (possibly negated) literals, where a literal can be an attribute-value pair or a Prolog literal. Typically, such conjunctions are constructed literal by literal. For instance, FOIL [16] selects the literal with maximal information gain. It is well-known that this leads to problems with literals introducing new variables, since these literals do not improve the class distribution unless they are combined with another literal which consumes the variable. There exist various attempts to solve this problem, e.g., look-ahead search which searches for more than one literal at a time. However, this approaches the problem in an ad-hoc fashion. The main issue is that in ILP it is unnatural to split literals which share variables other than the

one(s) occurring in the head of the rule. Such chunks of related literals are what we call first-order features.

Defining a rule body as consisting of features, where a feature is a conjunction of literals, also has the advantage of added expressiveness if we allow features to be negated. This would have the effect of allowing disjunctions of negated literals in the body. It also illustrates that body bias and feature bias are really two different things: e.g., we can allow negation of features but not of literals within features, or vice versa. The added expressiveness can be considerable. For instance, suppose that we define a body as a conjunction of possibly negated features, and a feature as a conjunction of possibly negated literals. The effect would be that any boolean combination of literals would be expressible in this combined body-feature bias.

3 Feature construction

So what are features, and how are they constructed? A natural choice – to which we will restrict ourselves in this paper – is to define a feature as a conjunction of (possibly negated) literals. Features describe subsets of the training set that are for some reason unusual or interesting. For instance, the class distribution among the instances described by the feature may be different from the class distribution over the complete training set in a statistically significant way. Alternatively, the feature may simply be shared by a sufficiently large fraction of the training set. In the first case, the feature is said to describe an *interesting subgroup* of the data, and several propositional and first-order systems exist that can discover such subgroups (e.g. Explora [8], MIDOS [19], and Tertius [6]). In the second case, the feature is said to describe a *frequent itemset*, and again several algorithms and systems exist to discover frequent itemsets (e.g. APRIORI [1] and WARMR [3]). Notice that the systems just mentioned are all discovery systems, which perform descriptive rather than predictive induction. Indeed, feature construction is a discovery task rather than a classification task.

3.1 The East-West challenge example

In this section we introduce a simple learning problem that will be used to illustrate different notions introduced in this paper. The learning task is to discover low size-complexity Prolog programs for classifying trains as Eastbound or Westbound [13]. The problem is illustrated in Figure 1. We illustrate two possible representations of this learning problem in Prolog.

Example 1 (East-West challenge in Prolog, non-flattened). Using a term-based (i.e. non-flattened) representation, the first train in Figure 1 can be represented as follows.

```
east([c(rect,short,single,no,2,l(circ,1)),
      c(rect,long, single,no,3,l(hexa,1)),
      c(rect,short,single,peak,2,l(tria,1)),
      c(rect,long, single,no,2,l(rect,3))]).
```

Here, a train is represented as a list of cars; a car is represented as a six-tuple indicating its shape, length, whether it has a double wall or not, its roof shape, number of wheels,

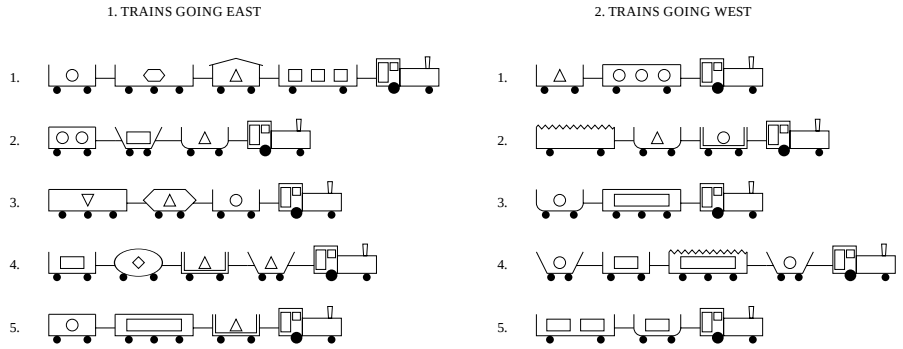

Fig. 1. The ten train East-West challenge.

and its load; finally, a load is represented by a pair indicating the shape of the load and the number of objects. A possible inductive hypothesis, stating that a train is eastbound if it has a short open car, is as follows:

```
east(T):-member(T,C),arg(2,C,short),arg(4,C,no)
```

Example 2 (East-West challenge in Prolog, flattened). A flattened representation of the same data, using function-free ground facts, is as follows.

```
east(t1).

hasCar(t1,c11).          hasCar(t1,c12).
cshape(c11,rect).        cshape(c12,rect).
clength(c11,short).      clength(c12,long).
cwall(c11,single).       cwall(c12,single).
croof(c11,no).           croof(c12,no).
cwheels(c11,2).          cwheels(c12,3).
hasLoad(c11,l11).        hasLoad(c12,l12).
lshape(l11,circ).        lshape(l12,hexa).
lnumber(l11,1).          lnumber(l12,1).

hasCar(t1,c13).          hasCar(t1,c14).
...                      ...
```

Using this representation, the above hypothesis would be written as

```
east(T):-hasCar(T,C),clength(C,short),croof(C,no)
```

Strictly speaking, the two representations in Examples 1 and 2 are not equivalent, since the order of the cars in the list in the first representation is disregarded in the second. This could be fixed by using the predicates `hasFirstCar(T,C)` and

`nextCar(C1,C2)` instead of `hasCar(T,C)`. For the moment, however, we stick to the above flattened representation; essentially, this means that we interpret a train as a *set* of cars, rather than a list. Under this assumption, and assuming that each car and each load is uniquely named, the two representations are equivalent. As a consequence, the two hypothesis representations are isomorphic: `hasCar(T,C)` corresponds to `member(T,C)`, and `clength(C,short)` corresponds to `arg(2,C,short)`. This isomorphism between flattened and non-flattened hypothesis languages is a feature of what we call *individual-centred* representations [7].

3.2 Identification of first-order features

In attribute-value learning, features only add expressiveness if the body bias allows negation of features. However, in first-order learning features occupy a central position. The distinguishing characteristic of first-order learning is the use of variables that are shared between some but not all literals. Namely, in the case that all variables occur in all literals, their only function is to distinguish literals occurring in rules from literals occurring in examples. In the latter case, by applying the Single Representation Trick, the variables can be dispensed with, thus reducing the rule to a propositional one.

Definition 1 (Global and local variables). *The variables occurring in the head of the rule are called* global variables*. Variables occurring only in the body are called* local variables*.*

Global variables are assumed to be universally quantified, and the scope of the universal quantifier is the rule. Local variables, called *existential variables* in logic programming terminology, are existentially quantified, and the scope of the existential quantifier is the rule body.

The key idea of first-order features is to restrict all interaction between local variables to literals occurring in the same feature. This is not really a restriction, as in some cases the whole body constitutes a single feature. However, often the body of a rule can be partitioned into separate parts which only share global variables.

Example 3. Consider the following Prolog rule, stating that a train is eastbound if it contains a short car and a closed car:

```
east(T):-hasCar(T,C1),clength(C1,short),
         hasCar(T,C2),not croof(C2,no)
```

The body of this clause consists of two features: `hasCar(T,C1),clength(C1,short)` or ‘has a short car’ and `hasCar(T,C2),not croof(C2,no)` or ‘has a closed car’. In contrast, the following rule

```
east(T):-hasCar(T,C),clength(C,short),not croof(C,no)
```

contains a single feature expressing the property ‘has a short closed car’.

It is easy to recognise the first-order features in any given rule, by focusing on the use of local variables.

Definition 2 (First-order features). *For any two body literals L_1 and L_2 of a given rule, L_1 and L_2 belong to the same equivalence class, $L_1 \sim_{lv} L_2$ iff they share a local variable. Clearly, $\sim_{lv}$ is reflexive and symmetric, and hence its transitive closure $=_{lv}$ is an equivalence relation inducing a partition on the body. The conjunction of literals in an equivalence class is called a* first-order feature.

We note the following simple result, which will be used later.

Proposition 1 (Propositionalising rules). *Let R be a Prolog rule, and let R' be constructed as follows. Replace each feature F in the body of R by a literal L consisting of a new predicate with R's global variable(s) as argument(s), and add a rule $L :- F$. R' together with the newly constructed rules is equivalent to R, in the sense that they have the same success set.*

Notice that R' contains only global variables, and therefore is essentially a propositional rule. R's first-order features have been confined to the background theory. Thus, provided we have a way to construct the necessary first-order features, *body construction in ILP is essentially propositional*.

By recognising the above property, an ILP problem can be transformed into a propositional learning problem, provided that appropriate first-order features can be constructed.

3.3 A declarative bias for first-order feature construction

Definition 2 suggests how to recognise features in a given clause, but it does not impose any restrictions on possible features and hence cannot be used as a declarative feature bias. In this section we define such a feature bias, following the term-based individual-centred representations introduced in [5] and further developed in [4]. Such representations collect all information about one individual in a single term, e.g., a list of 6-tuples as in the train representation of Example 1. In this example, the individual in question is a train, not its cars. Rules are formed by stating conditions on the whole term, e.g. `length(T,4)`, or by referring to one or more subterms and stating conditions on those subterms. Predicates which refer to subterms are called *structural predicates*: they come with the type of the term, e.g. list membership comes with lists, projections (n different ones) come with n-tuples, etc. Notice that projections are determinate, while list membership is not. In fact, the only place where non-determinacy can occur in individual-centred representations is in structural predicates.

Individual-centred representations can also occur in flattened form. In this case each of the individuals and most of its parts are named by constants, as in Example 2. It is still helpful to think of the flattened representation to be obtained from the term-based representation. Thus, `hasCar` corresponds to list/set membership and is non-determinate (i.e. one-to-many), while `hasLoad` corresponds to projection onto the sixth component of a tuple and thus is determinate (i.e. one-to-one). Keeping this correspondence in mind, the below definitions for the non-flattened case can be easily translated to the flattened case.

We assume a given type structure defining the type of the individual.

Definition 3 (Structural predicates). *Let τ be a given type signature, defining a single top-level type in terms of subtypes. A* structural predicate *is a binary predicate associated with a complex type in τ representing the mapping between that type and one of its subtypes. A* functional *structural predicate, or* structural function*, maps to a unique subterm, while a* non-determinate *structural predicate is non-functional.*

In general, we have a structural predicate or function associated with each non-atomic subtype in τ. In addition, we have *utility predicates* as in LINUS [10] (called *properties* in [4]) associated with each atomic subtype, and possibly also with non-atomic subtypes and with the top-level type (e.g., the class predicate). Utility predicates differ from structural predicates in that they do not introduce new variables.

Example 4. For the East-West challenge we use the following type signature. `train` is declared as the top-level set type representing an individual. The structural predicate `hasCar` non-deterministically selects a car from a train. `car` is defined as a 6-tuple. The first 5 components of each 6-tuple are atomic values, while the last component is a 2-tuple representing the load, selected by the structural function `hasLoad`. In addition, the type signature defines the following utility predicates (properties): `east` is a property of the top-level type `train`, the following utility predicates are used on the sub-type `car`: `cshape`, `clength`, `cwall`, `croof` and `cwheels`, whereas `lshape` and `lnumber` are properties of the sub-subtype `load`.

To summarise, structural predicates refer to parts of individuals (these are binary predicates representing a link between a complex type and one of its components; they are used to introduce new local variables into rules), whereas utility predicates present properties of individuals or their parts, represented by variables introduced so far (they do not introduce new variables). The language bias expressed by mode declarations used in other ILP learners such as Progol [14] or WARMR [3]) partly achieves the same goal by indicating which of the predicate arguments are input (denoting a variable already occurring in the hypothesis currently being constructed) and which are output arguments, possibly introducing a new local variable. However, mode declarations constitute a body bias rather than a feature bias.

Declarations of types, structural predicates and utility predicates define the feature bias. The actual first-order feature construction will be restricted by parameters that define the maximum number of literals constituting a feature, maximal number of variables, and the number of occurrences of individual predicates.

Definition 4 (First-order feature construction). *A* first-order feature *of an individual is constructed as a conjunction of structural predicates and utility predicates which is well-typed according to τ. Furthermore:*

1. *there is exactly one variable with type τ, which is free (i.e., not quantified) and will play the role of the global variable in rules;*
2. *each structural predicate introduces a new existentially quantified local variable, and uses either the global variable or one of the local variables introduced by other structural predicates;*
3. *utility predicates do not introduce new variables (this typically means that one of their arguments is required to be instantiated);*

4. *all variables are used either by a structural predicate or a utility predicate.*

The following first-order feature could be constructed in the above feature bias, allowing for 4 literals and 3 variables:

```
hasCar(T,C),hasLoad(C,L),lshape(L,tria)
```

4 Application: learning nondeterminate clauses with LINUS

In the previous sections we have argued that feature construction is a crucial notion in inductive rule learning. We have given precise definitions of features in first-order languages such as Prolog. First-order features bound the scope of local variables, and hence constructing bodies from features is essentially a propositional process that can be solved by a propositional rule learner such as CN2. In this section we show the usefulness of this approach by solving two non-determinate ILP tasks with the transformation-based rule learner LINUS [10].

We provide LINUS with features defining background predicates, as suggested by Proposition 1. For instance, in the trains example we add clauses of the following form to the background knowledge:

```
train42(T):-
  hasCar(T,C),hasLoad(C,L),lshape(L,tria)
```

LINUS would then use the literal `train42(T)` in its hypotheses. Such literals represent propositional properties of the individual.

In the two experiments reported on in this section we simply provide LINUS with all features that can be generated within a given feature bias (recall that such a feature bias includes bounds on the number of literals and first-order features). Alternatively, we can first apply a relevancy filter [12] to eliminate irrelevant features; or we can use a descriptive learner such as Tertius [6] to generate only features that correlate sufficiently with the class attribute. We are currently working on such methods for non-exhaustive feature construction; some preliminary results are mentioned in the next section.

Experiment 1 (LINUS applied to the East-West challenge) *We ran LINUS on the ten trains in Figure 1, using a non-determinate background theory consisting of all 190 first-order features with up to two utility predicates and up to two local variables. Using CN2, the following rules were found:*

```
east(T):-
  hasCar(T,C1),hasLoad(C1,L1),lshape(L1,tria),lnumber(L1,1),
  not (hasCar(T,C2),clength(C2,long),croof(C2,jagged)),
  not (hasCar(T,C3),hasLoad(C3,L3),clength(C3,long),lshape(L3,circ)).
west(T):-
  not (hasCar(T,C1),cshape(C1,ellipse)),
  not (hasCar(T,C2),clength(C2,short),croof(C2,flat)),
  not (hasCar(T,C3),croof(C3,peak),cwheels(C3,2)).
```

These rules were found allowing negation in the body, but not within features. If negation is also allowed within the feature bias, the following simple rules are induced:

```
east(T):-
  hasCar(T,C),clength(C,short),not croof(C,no).
west(T):-
  not (hasCar(T,C),clength(C,short),not croof(C,no)).
```

That is, a train is eastbound if and only if it has a short closed car.

The mutagenesis learning task [15] concerns predicting which molecular compounds cause DNA mutations. The mutagenesis dataset consists of 230 classified molecules; 188 of these have been found to be amenable to regression modelling, and the remaining 42, to which we restrict attention here, as 'regression-unfriendly'. The dataset furthermore includes two hand-crafted indicator attributes I_1 and I_a to introduce some degree of structural detail into the regression equation; following some experiments in [15] we did not include these indicators.

Experiment 2 (LINUS applied to mutagenesis) *We ran LINUS on the 42 regression-unfriendly molecules, using a non-determinate background theory consisting of all 57 first-order features with one utility literal concerning atoms (i.e., discarding bond information). Using CN2, the following rules were found:*

```
mutag(M,false):-not (has_atom(M,A),atom_type(A,21)),
                 logP(M,L),between(1.99,L,5.64).
mutag(M,false):-not (has_atom(M,A),atom_type(A,195)),
                 lumo(M,Lu),between(-1.74,Lu,-0.83),
                 logP(M,L),L>1.81.
mutag(M,false):-lumo(M,Lu),Lu>-0.77.

mutag(M,true):-has_atom(M,A),atom_type(A,21),
               lumo(M,Lu),Lu<-1.21.
mutag(M,true):-logP(M,L),between(5.64,L,6.36).
mutag(M,true):-lumo(M,Lu),Lu>-0.95,
               logP(M,L),L<2.21.
```

Three out of 6 clauses contain first-order features. Notice how two of these concern the same first-order feature 'having an atom of type 21' – incidentally, such an atom also features in the (single) rule found by Progol on the same dataset. Running CN2 with only the lumo and logP attributes produced 8 rules; thus, this experiment suggests that first-order features can enhance the understandability of learned rules. Furthermore, we also achieved higher predictive accuracy: 83% with first-order features (as opposed to 76% using only lumo and logP). This accuracy is the same as achieved by Progol, having access to bond information and further structural background knowledge.

5 Conclusions and further work

This work points out that there is a tradeoff between how much effort a learner puts in the following steps of the hypothesis generation process: rule construction, body construction, and feature construction. In this work we have shown that by devoting enough effort to feature construction, even complex relational learning tasks can be solved by simple propositional rule learning systems.

In propositional learning, the idea of augmenting an existing set of attributes with new ones is known under the term constructive induction. A first-order counterpart of constructive induction is predicate invention. This work is in the mid-way: we perform

a simple form of predicate invention through first-order feature construction, and use the constructed features for propositional learning.

Other researchers have taken a similar mid-way approach to the propositionalisation of relation learning problems. In the East-West challenge problem, Peter Turney's RL-ICET algorithm [18] achieved one of the best results due to exhaustive propositional feature construction, introducing new propositional features for all combinations of up to three first-order literals. The approach taken by Kramer et al. [9] is more sophisticated. It performs predicate invention through stochastic search of best features of variable length. A fitness function optimising description length (similar to Turney's costs of features) as well as other criteria (generality/specificity and variability) are used to guide the search of best features. Srinivasan and King [17] perform propositionalisation through predicate invention achieved by a variety of predictive learning techniques, including linear regression, rule and decision tree learning, and inductive logic programming.

In this paper we have also shown that the traditional limitations of transformation-based approaches such as LINUS (which does not allow local variables) and its successor DINUS (which allows only determinate local variables) can be alleviated by means of non-determinate feature construction. While in this work feature construction was exhaustive within a given feature bias, in future work we intend to study two non-exhaustive approaches. The first is to use a filter for eliminating irrelevant features, as done in the work by Lavrač and Gamberger [12]. We did some preliminary experiments in the trains domain using two generated feature sets: the above-mentioned set containing 190 features and a larger set of 564 features. In both cases, the redundancy eliminated all except for 16 relevant features. One may further reduce the set of relevant features by selecting a quasi-minimal set of features needed for hypothesis construction, as proposed by the same authors in [11]. This resulted in a minimal set of three features only. As an alternative approach, we plan to use a descriptive learner such as Tertius [6] for constructing features that correlate significantly with the class attribute.

Acknowledgements

We are grateful to Nicolas Lachiche for implementing feature construction. This work was partially supported by a Joint Project with Central/Eastern Europe funded by the British Royal Society, the Slovenian Ministry of Science and Technology, and the IST-1999-11495 project *Data Mining and Decision Support for Business Competitiveness: A European Virtual Enterprise* funded by the European Union.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In U.M. Fayyad, G. Piatetski-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, 1996.
2. P. Clark and T. Niblett. The cn2 induction algorithm. *Machine Learning*, 3:261–284, 1989.
3. L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.

4. P. Flach and N. Lachiche. 1BC: A first-order Bayesian classifier. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 92–103. Springer-Verlag, 1999.
5. P.A. Flach, C. Giraud-Carrier, and J.W. Lloyd. Strongly typed inductive concept learning. In D. Page, editor, *Proceedings of the 8th International Conference on Inductive Logic Programming*, volume 1446 of *Lecture Notes in Artificial Intelligence*, pages 185–194. Springer-Verlag, 1998.
6. Peter Flach and Nicolas Lachiche. Confirmation-guided discovery of first-order rules with Tertius. *Machine Learning*, 2000. Forthcoming.
7. Peter A. Flach. Knowledge representation for inductive learning. In Anthony Hunter and Simon Parsons, editors, *Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU'99)*, volume 1638 of *Lecture Notes in Artificial Intelligence*, pages 160–167. Springer-Verlag, July 1999.
8. W. Klösgen. Explora: A multipattern and multistrategy discovery assistant. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 249–271. AAAI Press, 1996.
9. S. Kramer, B. Pfahringer, and C. Helma. Stochastic propositionalization of non-determinate background knowledge. In D. Page, editor, *Proceedings of the 8th International Conference on Inductive Logic Programming*, volume 1446 of *Lecture Notes in Artificial Intelligence*, pages 80–94. Springer-Verlag, 1998.
10. N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
11. N. Lavrač, D. Gamberger, and S. Džeroski. An approach to dimensionality reduction in learning from deductive databases. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 337–354. Department of Computer Science, Katholieke Universiteit Leuven, 1995.
12. Nada Lavrač, Dragan Gamberger, and Peter Turney. A relevancy filter for constructive induction. *IEEE Intelligent Systems*, 13(2):50–56, March-April 1998.
13. D. Michie, S. Muggleton, D. Page, and A. Srinivasan. To the international computing community: A new East-West challenge. Technical report, Oxford University Computing laboratory, Oxford,UK, 1994.
14. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
15. S. Muggleton, A. Srinivasan, R. King, and M. Sternberg. Biochemical knowledge discovery using Inductive Logic Programming. In H. Motoda, editor, *Proceedings of the first Conference on Discovery Science*, Berlin, 1998. Springer-Verlag.
16. J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.
17. A. Srinivasan and R.D. King. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, volume 1314 of *Lecture Notes in Artificial Intelligence*, pages 89–104. Springer-Verlag, 1996.
18. P. Turney. Low size-complexity inductive logic programming: The East-West challenge considered as a problem in cost-sensitive classification. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 308–321. IOS Press, 1996.
19. S. Wrobel. An algorithm for multi-relational discovery of subgroups. In J. Komorowski and J. Zytkow, editors, *Proceedings of the 1st European Symposium on Principles of Data Mining and Knowledge Discovery*. Springer-Verlag, 1997.