Advances in Inductive Logic Programming
L. De Raedt (Ed.)
IOS Press, 1996

Handling Imperfect Data in Inductive Logic Programming

Nada Lavrač (1), Sašo Džeroski (1,2), Ivan Bratko (1,3)

(1) Jožef Stefan Institute, Jamova 39, 61111 Ljubljana, Slovenia

(2) German National Research Center for Information Technology
Schloss Birlinghoven, 53745 Sankt Augustin, Germany

(3) Faculty of Electrical Engineering and Computer Science
University of Ljubljana, Tržaška 25, 61111 Ljubljana, Slovenia

Abstract. Early Inductive Logic Programming (ILP) systems were mostly interactive and assumed correct data. With the advent of empirical ILP the interest in handling imperfect data in ILP has increased. Many recent empirical ILP systems include elaborate mechanisms for handling imperfect data, making ILP applicable to real-life problems. Advances in handling imperfect data include the development of new heuristics, advanced search techniques, as well as techniques for handling real-valued data. The paper gives a survey of these advances with a special emphasis on the developments in the ESPRIT III Project 6020 Inductive Logic Programming.

1 Introduction

Early Inductive Logic Programming (ILP) systems were mostly interactive. Interactive ILP systems typically learn definitions of multiple predicates from a small set of examples and queries to the user. Interactive ILP systems such as MIS [57], MARVIN [55], CIGOL [44] and CLINT [8] assume correct data and do not address the problem of handling imperfect data.

With the advent of empirical ILP, which has a greater potential for practical applications, the interest in handling imperfect data in ILP has increased. Empirical ILP systems typically learn a definition of a single predicate from a larger collection of examples. Representatives of empirical ILP systems which include mechanisms for handling imperfect data are FOIL [53], mFOIL [12], LINUS [37], GOLEM [45] and PROGOL [43]. Techniques for handling imperfect data have made empirical ILP applicable to real-life problems [41, 34, 1].

Both empirical and interactive ILP [8] are situated within the so-called 'normal' ILP framework. A new approach to ILP has been recently developed, called 'nonmonotonic' ILP [9, 11]. Techniques for handling imperfect data within nonmonotonic ILP have also been developed.

The paper gives a survey of state-of-the-art techniques for handling imperfect data, with a special emphasis on the techniques developed within the ESPRIT III Project 6020 Inductive Logic Programming. The paper is organised as follows. It first gives

an overview of the different kinds of imperfect data (Section 2) and then presents the related noise-handling mechanisms, i.e., heuristics (Section 3) and techniques for handling missing values and too sparse training examples (Section 4). Advanced search techniques are presented in Section 5. In Section 6, techniques for handling real-valued data are outlined. Section 7 gives a summary and directions for further research.

2 Imperfect data

In an ideal situation, the induced concept description (a hypothesis H) will 'agree' with the classifications of the descriptions of all concept instances (training examples E). However, in practice it frequently happens that data given to the learner contains various kinds of errors, either random or systematic. In real-life problem domains, noise in data is often due to imprecise devices, imperfect measuring equipment, erroneous transmission of data, as well as systematic errors caused by incorrect calibration of the measuring equipment (for example, the readings of the measuring equipment may be consistently too low). In such circumstances, we say that the learner has to deal with *imperfect data.* To allow it to do so, the requirement that H should correctly classify all examples in E needs to be relaxed. A desirable property of a learner is then to avoid the effects of imperfect data by distinguishing between genuine regularities in the examples and regularities due to chance or error.

In ILP, where relational descriptions are learned in a first-order language from examples E and background knowledge B, the following forms of imperfect data can be encountered [34]:

1. random errors (*noise*):
   (a) noise in training examples E (caused by erroneous argument values and/or erroneous classification of facts as true or false),
   (b) noise in background knowledge B;
2. too sparse training examples E (*incompleteness*) from which it is difficult to reliably detect regularities;
3. imperfect background knowledge B (*inappropriateness*):
   (a) background knowledge B may contain predicates that are not relevant for the learning task,
   (b) predicates in B may be insufficient for learning (essential predicates may be missing);
4. missing argument values in training examples E (*missing values*).

Learning systems usually have a single mechanism for dealing with the first three kinds of imperfect data, i.e., with noisy, incomplete and inappropriate data. Such mechanisms, often called *noise-handling mechanisms*, are typically designed to prevent the induced hypothesis from overfitting the data set, i.e., to avoid overly specific concept descriptions with low classification accuracy on unseen instances. Missing values are usually handled by a separate mechanism.

An additional type of data which is common in practice is real-valued data. Regularities to be found in such data may involve equations, inequalities and real-valued

constants that have to be determined by the learning system. While real-valued data is very often imperfect, this need not be the case in principle. However, it seems that handling real-valued data requires the use of predicates with special semantics, thus eluding the elegance of other ILP techniques. This overview includes also methods for handling real-valued data.

3 Heuristics for handling imperfect data

Noise-handling mechanisms can be incorporated in search heuristics and in stopping criteria used in hypothesis construction. Moreover, hypotheses fulfilling stopping criteria may still be evaluated according to some quality measure, giving a preferencial order among hypotheses. In addition, induced hypotheses can be subjected to some form of post-processing.

In this section, heuristics are studied in the framework of empirical ILP systems, employing the covering approach for hypothesis construction, and the top-down search of refinement graphs for the construction of a single clause. Hypothesis construction starts with an empty hypothesis H and the initial training set E. The covering algorithm basically consists of three steps:

1. construct a clause c
2. construct new hypothesis H' by adding c to the current hypothesis H
3. construct new training set E' by removing from the current training set E the positive examples covered by c.

The steps of the covering algorithm are usually repeated until the set of yet uncovered training examples becomes empty.

Clause construction is implemented by applying a refinement operator ρ which takes the current clause $c = T \leftarrow Q$ (T is an atom and Q is a conjunction of literals) and builds a set of its specializations (refinements): $\rho(c) = \{T \leftarrow Q' \mid Q' = Q, L\}$.

Each specialization $c' \in \rho(c)$ is built by conjunctively adding a literal L to the body Q of c: $c' = T \leftarrow Q, L$ (comma stands for conjunction). Specialization starts with the clause c which has the head $T = p(X_1, \ldots, X_n)$ (p is the predicate symbol of the target relation to be learned) and an empty body: $c = T \leftarrow$.

The algorithm keeps one (or beam-size of the) 'best' clause(s) and replaces the best clause with its 'best' refinement, until the stopping criterion for clause construction is satisfied which decides when to stop adding literals to a clause. In domains with perfect data, this criterion requires consistency (no covered negative examples). In the hypothesis construction (covering) loop, another stopping criterion decides when to stop adding clauses to the hypothesis. In domains with perfect data, this criterion requires completeness (all positive examples covered).

In order to find the best clause c and the best hypothesis H, a quality measure needs to be defined. The *quality* $f(c)$ or $f(H)$ with respect to the current training set E is intended to reflect how well the clause/hypothesis 'fits' the positive training examples in E, or how 'good' is the distribution (split) of positive and negative examples covered by the clause. Generally speaking, the more positive examples covered the better; the more negative examples covered, the worse.

3.1 Search heuristics

This section outlines some heuristics used to guide the search for the best clause. These heuristics are able of dealing with noise in the training examples (see item 1.a, Section 2) and too sparse training examples (item 2).

Let n be the number of examples in the initial training set, $n^{\oplus}$ of which are positive and $n^{\ominus}$ negative. Let $n(c)$ be the number of examples in the current training set E which are covered by the clause $c = T \leftarrow Q$, $n^{\oplus}(c)$ positive and $n^{\ominus}(c)$ negative. The heuristics are concerned with finding the clause which 'best fits' the data (maximizing the set of positive examples covered), or a clause with the 'best split' between $n^{\oplus}(c)$ positive and $n^{\ominus}(c)$ negative examples covered by c. The simplest heuristic is the expected *accuracy* of a clause:

$$A(c) = p(\oplus|c)$$

The expected accuracy is defined as the probability that an example covered by clause c is positive. The greater the probability, the better. *Informativity* (measured in bits) is another frequently used heuristic. It is defined as:

$$I(c) = -log_2\ p(\oplus|c)$$

The above two heuristics that estimate the quality of a clause can be used as search heuristics (the best clause maximizes accuracy and minimizes informativity), as stopping criteria in clause construction (stop clause construction when some threshold is reached) as well as preference criteria for comparing candidate clauses.

The so-called 'gain' heuristics, on the other hand, estimate the quality of a literal L when added to the clause in one refinement step, rather than estimating the clause itself. Gain functions $f(L) = f(c', c)$ are based on the difference in clause quality when a clause c is substituted with its refinement c'. Gain heuristics are only used as search heuristics. They include *accuracy gain* $AG(c', c) = A(c') - A(c)$ and *information gain* $IG(c', c) = I(c) - I(c')$, as well as *weighted accuracy gain* $WAG(c', c) = \frac{n^{\oplus}(c')}{n^{\oplus}(c)}\ AG(c', c)$ and *weighted information gain* $WIG(c', c) = \frac{n^{\oplus}(c')}{n^{\oplus}(c)}\ IG(c', c)$.

Different probability estimates can be used to compute $p(\oplus|c)$, the simplest (but not the most appropriate) being the *relative frequency* of covered positive examples

$$p(\oplus|c) = \frac{n^{\oplus}(c)}{n(c)}$$

More appropriate probability estimates are the *Laplace estimate* [48] defined as $p(\oplus|c) = \frac{n^{\oplus}(c)+1}{n(c)+2}$, [1] and in particular the *m-estimate* [4]:

$$p(\oplus|c) = \frac{n^{\oplus}(c) + m\ .\ p_a(\oplus)}{n(c) + m}$$

The m-estimate takes into account the prior probability of class $\oplus$, estimated by the relative frequency of positive examples in the initial training set: $p_a(\oplus) = \frac{n^{\oplus}}{n}$. The parameter m is set subjectively, according to the expected amount of noise in the examples (larger m for more noise).

An overview, analysis and evaluation of the above heuristics for handling noise in ILP are given in [34]. An empirical evaluation of the above heuristics and probability

[1] In ILP there are two classes: $\oplus$ and $\ominus$. In general, for distinguishing one class $\oplus$ among k different classes, the Laplace estimate is computed as follows: $p(\oplus|c) = \frac{n^{\oplus}(c)+1}{n(c)+k}$.

estimates is given also in [36]. The empirical study shows that, with respect to noise handling, $A(c)$ and $I(c)$ are less appropriate than $IG(c',c)$ and $AG(c',c)$. $IG_m(c',c)$ and $AG_m(c',c)$ give similar performance as $WIG_{rf}(c',c)$, $WIG_{Lap}(c',c)$ and $WAG_{rf}(c',c)$, $WAC_{Lap}(c',c)$, where rf, Lap and m stand for relative frequency, Laplace and m-estimate, respectively. The best performance in noise handling is achieved when using $WIG_m(c',c)$ and $WAG_m(c',c)$.

A case study of noise handling by LINUS [37] and FOIL [53] is described in [33, 15]. FOIL uses a variant of weighted information gain as a search heuristic, whereas LINUS uses search heuristics of the propositional learning algorithms ASSISTANT [3] and CN2 [6]. In addition, an accuracy-based heuristic is implemented in LINUS in the post-processing of hypotheses. In mFOIL [12], an accuracy-based heuristic is used as a search heuristic and a CN2-like criterion of significance [6] is used as a stopping criterion in clause construction. In FOIL, the stopping criterion is based on the encoding length restriction. In [33] it was shown that the stopping criterion of FOIL has disadvantages for noise-handling. Other approaches based on the encoding length will be discussed in Section 3.2.

The ICL [11] and CLAUDIEN [9] systems, which operate in the nonmonotonic ILP framework, adopt the view that examples are interpretations (sets of ground facts) instead of ground facts. Having adopted this view, one can employ the same heuristics as in propositional learning algorithms: the number of examples covered is in this case the number of interpretations in which the theory (hypothesis) is true. This idea is implemented in ICL, which uses the same heuristics as CN2, but counts interpretations as examples.

In the study of multiple predicate learning [10], it was shown that coverage in most ILP systems means *extensional coverage*. Systems employing extensional coverage consider the examples covered by the current clause c independently, i.e., not in the context of other clauses in the hypothesis H. When learning a recursive predicate definitions, say for predicate p_i, *intensional coverage* needs to be used which takes into account the examples covered by c in the context of the hypothesis H_i. Moreover, in multiple predicate learning, *global coverage* w.r.t. the entire hypothesis $H = H_1 \cup \ldots \cup H_k$ (for k different predicates) needs to be considered, as opposed to *local coverage* in the context of the predicate definition H_i.

Besides the accuracy and informativity based heuristics, a search heuristic used in the propositional learner RELIEF [27] was adapted to ILP. The key idea of the RELIEF heuristic is to find the best split among 'near' instances only. Thus, instead of estimating the quality of split between $n^{\oplus}(c)$ positive and $n^{\ominus}(c)$ negative examples covered by the clause among all instances in the current training set E, the RELIEF heuristic estimates the quality of split only among instances that are 'near' each other. The ILP system ILP-R [51, 52] takes into account k nearest hits (nearest neighbours from the class $\oplus$) and k nearest misses (nearest neighbours from class $\ominus$) when estimating the quality of split. In addition, a method for heuristically evaluating non-determinate literals has been developed. In ILP-R, the RELIEF-based search heuristic is used in a beam search covering algorithm. It was shown that using the RELIEF-based heuristic, ILP-R was able to detect literal dependencies, which is an unsolvable problem for standard greedy search heuristics. Some other features of the ILP-R algorithm are described in Section 5.

3.2 Heuristic quality measures: significance and compression

This section studies heuristic quality measures that enable the gradation of induced hypotheses w.r.t. their significance and compression. These heuristics may also be used for clause evaluation.

For clause evaluation, a measure of *significance* used in the propositional learner CN2 [6] has been adapted to ILP and used in the ILP systems mFOIL [12] and ICL [11] as a stopping criterion in clause construction.

The significance test is based on the likelihood ratio statistic [23]. If a clause c covers $n(c)$ examples, $n^{\oplus}(c)$ of which are positive, the value of the statistic can be calculated as follows. Let $p_a(\oplus)$ and $p_a(\ominus)$ be the prior probabilities of classes $\oplus$ and $\ominus$, estimated by the relative frequencies of positive and negative examples in the entire training set: $p_a(\oplus) = \frac{n^{\oplus}}{n}$ and $p_a(\ominus) = \frac{n^{\ominus}}{n}$. In addition, $p(\oplus|c) = \frac{n^{\oplus}(c)}{n(c)}$ is the expected accuracy defined as the probability that an example covered by clause c is positive, and $p(\ominus|c) = 1 - p(\oplus|c)$. Then the likelihood ratio estimate of clause c, $LR(c)$, is computed as follows:

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

This statistic is distributed approximately as χ^2 with one degree of freedom. If its value is above a specified significance threshold, the clause is deemed significant. In mFOIL, clause refinement stops when significance drops under the predefined significance threshold. The search for clauses is terminated when too few positive examples (possibly zero) are left for a generated clause to be significant or when no significant clause can be found with expected accuracy greater than the default.

Demanding the expected accuracy (estimated by the heuristic value) to be higher than the default accuracy may cause constructing no clauses at all in domains where negative examples prevail. In such cases, the criterion of accuracy may be ommited and the expected accuracy of the constructed clauses disregarded, provided that clauses are significant; the search then terminates when no further significant clause can be constructed. Afterwards, the clauses with very low accuracy might be discarded. This approach was applied in experiments with mFOIL on the finite element mesh design domain [12].

In mFOIL, examples are ground facts of the target predicate and therefore ground facts covered and not covered by c are counted when computing $LR(c)$. In ICL, on the other hand, examples are interpretations (sets of ground facts), and interpretations covered and not covered are counted [11]. A different notion of coverage is used as compared to mFOIL: an interpretation is covered by a clause if the clause is true in the interpretation.

Compression measures [40, 47, 53] are theoretically based on the Minimum Description Length (MDL) principle of Rissanen [54] and Chaitin [5]. As opposed to measures described in Section 3.1 which try to find a hypothesis which best 'fits' the example set, compression measures integrate a measure of complexity (simplicity or understandability) and correctness (accuracy or quality of fit) into one measure. This feature, as well as the fact that the MDL principle provides a theoretically justified basis for grading candidate hypotheses makes compression heuristics particularly well suited for ILP. Although the measures are intended to deal with imperfect training examples E as well as imperfect background knowledge B (items 1, 2 and 3, Section 2), an in-depth study in handling noisy background knowledge is not yet available.

Let theory $T = H \cup B$, where H is the induced hypothesis and B stands for the background theory. The posterior probability $p(T|E)$ of theory T given the examples E, that needs to be maximized, can be computed using Bayes' law:

$$p(T|E) = \frac{p(T) \cdot p(E|T)}{p(E)}$$

Probability $p(E|T)$ can be interpreted as the accuracy of T w.r.t. E, i.e., the degree to which T fits the examples. Probability $p(E)$ can be assumed to be constant (equal to 1, as the examples are assumed to be correct and examples are independent of the choice of T).

The key idea underlying compression measures goes as follows. The *informativity* or *information content* of a theory T, defined as $I(T) = -log_2\ p(T)$ (see also Section 3.1), equals the minimum description length $K(T)$ of theory T using an optimal encoding (the so-called Kolmogorov complexity of the theory). According to Bayes' law, rewritten for informativity, $I(T|E) = I(T)+I(E|T)$.[2] Maximizing $p(T|E)$ is equivalent to minimizing $I(T|E)$.

Theory T is *compressive* if $I(T) + I(E|T) < I(E \cup B)$, i.e., if the message length to describe the theory $T = H \cup B$ and the message length to describe the examples given the theory T requires less bits than the length of the encoding of the original examples E and background theory B. The most compressive theory is the one for which $I(T|E) = I(T) + I(E|T)$ is minimal.

Since the optimal encoding is not computable, computable approximations need to be used. Various encoding schemes have been proposed [40, 53, 7, 47]. The encoding scheme proposed in [47], implemented in the ILP system GOLEM [45], takes into account the information necessary to encode the background knowledge as well as the proofs which would generate the training examples covered. Good results in practical domains have been reported using this scheme [47].

Some shortcomings of the MDL principle implementation of [47] have been identified in [28]. It was observed that the method performs poorly if the size of the training set is small (since it oversimplifies the hypothesis for small sample sizes). The improved method implemented in the ILP system MILP [29, 30] can handle small training sets and also inconsistent hypotheses, since it chooses the most accurate among the compressive hypotheses. In this case, the MDL principle is used as a significance test: a compressive hypothesis is considered significant. Experiments (in the KRK domain) have shown that the improved compression heuristic outperforms the original implementation of [47].

A recent study of MDL-based heuristics [60] gives an overview of current MDL implementations in ILP. It identifies an important misinterpretation of the MDL principle as used in existing ILP compression measures and proposes a solution through restricting $I(E|T)$ to positive examples only. It is argued that the proposed measure may be used for controlling predicate invention.

4 Handling missing values and too sparse training examples

Let us first describe two simplistic mechanisms for handling missing values (item 4, Section 2.1). The simplest way is to replace a missing ('don't know') value by the majority value of the attribute/argument, where majority is considered only within

[2] Notice that $I(E|T)$ can be interpreted as the length (in bits) of the proofs of the examples E given T [47]. An alternative model-based view is offered by other authors, e.g., [7].

the class ($\oplus$ or $\ominus$) of the training example. A more sophisticated way is to replace an example with a missing value by several examples, one for each of the possible attribute/argument values, weighted by the conditional (with regard to the class of the example) probabilities of the values. The above two mechanisms are implemented in the ILP system LINUS [34].

A different approach, applicable in the case of missing classification values of examples in E, solves this problem by combining learning and conceptual clustering techniques. The approach is implemented in the system COLA [21]. The system assumes the following problem setting: given is a small set of classified training instances (a set of sparse training examples) and a set of unclassified instances (examples with missing class values $\oplus$ or $\ominus$). COLA uses a conceptual clustering algorithm KBG [2] on the entire set of instances. It climbs the hierarchy tree and uses the classified instances (by applying the m-estimate) to identify (single or disjunctive) class descriptions forming H. The approach shows that by combining learning with clustering, the sparseness of E and the missing classification values in E can well be dealt with. It was shown that in the finite element mesh design problem the classification accuracy of H increased: a more general hypothesis was induced given few positive examples, and a more specific one given few negative examples. A similar approach to dealing with too sparse examples is taken in the system RIBL [22], which is an adaptation of instance-based (nearest neighbor) approaches to a relational representation.

5 Advanced search techniques

One of the key problems in ILP is how to search the space of logic programs efficiently. Existing ILP algorithms use variants of greedy search to explore the space of logic programs. Due to myopic behavior, a greedy search algorithm is easily trapped in a local minimum. To alleviate the problem of myopic greedy search, several advanced search techniques have been developed: stochastic search, genetic search, exhaustive beam search and bi-directional search techniques.

5.1 Stochastic search

Greedy search may be replaced by stochastic search, similar to the search performed in a Markovian neural network [31]. A stochastic search algorithm called sFOIL was first developed [50], followed by a more advanced stochastic search algorithm MILP [30], based on ideas from simulated annealing. Stochastic ILP algorithms use stochastic search in the state space of logic programs. Successors of the currently investigated state (program) are obtained by a MIS refinement operator. Transitions in the state space are probabilistic. The probability of a transition to a state is proportional to the quality of the state, where the quality is evaluated using a combined MDL-accuracy heuristic (described in Section 3.2). MILP achieved good results when tested in the KRK chess end-game domain and the finite element mesh design problem.

5.2 Genetic search

An ILP system GILP [63, 64] based on genetic algorithms was developed. Genetic algorithms operate on a population, i.e., a set of candidate solutions, called chromo-

somes. In GILP, chromosomes are Prolog clauses represented as binary strings. In GILP, the genetic algorithm operating on a population of Prolog clauses is employed in conjunction with the covering principle until a program evolves which maximizes a given heuristic measure of quality. In clause construction, a genetic algorithm finds a clause which maximizes the coverage of positive and minimizes the coverage of negative examples. If the clause satisfies the significance test, it is added to the current hypothesis. In clause construction, the search space is thus not structured according to generalization/specialization. Instead, recombination of concepts is used together with a stochastic search strategy of the genetic algorithm, thus introducing a novel search bias into ILP. To facilitate the recombination, a stack-based representation of hypotheses was developed. GILP was able to synthesize logic programs in several benchmark domains and achieved results comparable to those of FOIL and GOLEM.

5.3 Exhaustive beam search

An ILP approach which uses a RELIEF-based search heuristic, exhaustive beam search and a novel declarative bias has been developed and implemented in the ILP-R system [51]. The RELIEF-based search heuristic [27] (described in Section 3.1) is used in a beam search covering algorithm. At each individual step, the beam of best literals (best with respect to the 'topology' of examples - k nearest hits and k nearest misses) is exhaustively searched in order to find the most informative 'phrase', i.e., a conjunction of literals (and not a single literal, as is usually the case in ILP systems). In addition, the proposed method employs a novel declarative bias which keeps the growth of the training set within linear boundaries (with respect to the clause length). Finally, in order to improve classification, a naive Bayesian classifier has been incorporated into the ILP-R learning system [52]. Experimental results have shown that when classifying unseen instances the combination of ILP-R with the naive Bayesian classifier sometimes significantly improves both the classification accuracy and the average information score.

5.4 Bi-directional search

A form of bi-directional search is used in the PROGOL algorithm [43], combining the advantages of top-down and bottom-up search. The algorithm proceeds as follows: (1) it randomly selects an example e_i, (2) uses inverse resolution to construct the most specific clause $c(e_i, B)$ that explains e_i (such that $c(e_i, B) \vdash e_i$), (3) by top-down search of the subsumption lattice finds a clause c_i which subsumes $c(e_i, B)$, is consistent w.r.t. the current example set E and maximally compresses a set of examples subsumed by c_i, (4) adds c_i to the current hypothesis H, and repeats the process with the examples that are still not covered until no more compression is possible. A simple MDL-based compression measure is used in PROGOL. The algorithm results in a substantial reduction of the search space (this improves efficiency w.r.t. FOIL) and has no determinacy restriction (as compared to GOLEM). PROGOL has already shown its great application potential in molecular biology, in particular in predicting mutagenicity [58, 59].

6 Handling real-valued data

Practical ILP systems must be able to deal with real-valued data, particularly if applied as tools for knowledge discovery in databases or tools for modeling dynamic systems. Several approaches to handling real-valued data have been developed for both the normal and the nonmonotonic ILP semantics. The simplest form of handling real-valued data is to allow inequalities that compare a real-valued variable to a real number. Several ILP systems now have number-handling capabilities of this kind, including FOIL [53] and PROGOL [43].

More complex forms of handling real-valued data include the use of linear or nonlinear equations and consequently the use of regression. In normal ILP, this has been implemented in the relational regression [17, 25, 26] and qualitative regression [46] approaches, as well as in Inductive Constraint Logic Programming [56]. In nonmonotonic ILP, this has been implemented in the approach to discovering models of dynamic systems [16, 18] and the approach to discovering clausal constraints [13, 61].

6.1 Relational regression in normal ILP

The relational regression task can be defined as follows: *Given* (1) training examples as positive ground facts for the target predicate $r(Y, X_1, ..., X_n)$, where the variable Y has real values, and (2) background knowledge predicate definitions, *find* a definition for $r(Y, X_1, ..., X_n)$, such that each clause has a literal binding Y (assuming that $X_1, ..., X_n$ are bound). Typical background knowledge predicates include $less\_or\_equal(A, B)$, $plus(A, B, C)$, $minus(A, B, C)$, and $multiply(A, B, C)$. A transformation approach [17] and a direct approach [25] to relational regression have been developed. Both include techniques for handling noisy data, such as pre-pruning and post-pruning.

The transformation approach [17] is implemented within the DINUS algorithm [35]. DINUS inherits the ability to handle imperfect and real-valued data from the underlying propositional systems. In DINUS, background knowledge predicates are used to form propositional features, following the introduction of determinate new variables. A propositional regression algorithm RETIS [24] is used for learning. The obtained propositional description is then transformed back to relational form. This approach was illustrated with an example from behavioural cloning [20], where the goal is to learn control rules for a dynamic system, given example traces of successful control actions. It was also tested on the problem of predicting workpiece surface roughness for a grinding process.

A direct approach to relational regression is implemented in the system FORS (First Order Regression System) [25, 26], which is based on FOIL. FORS performs top-down search of a refinement graph, starting with the initial clause $r(Y, X_1, ..., X_n) \leftarrow$. Literals of the form $A = v$ (for discrete attributes), $A \leq v$ and $A \geq v$ (for continuous real-valued attributes), and literals that use background knowledge predicates may be added to the clause body. For example, literals of the form $less\_than(A, B)$ may be added, provided that this predicate is in the background knowledge. In each clause, FORS can predict a value for the target variable Y as the output value of a background knowledge literal, as a constant, or as a linear combination of variables appearing in the clause (using linear regression). FORS employs a covering approach, beam search, a heuristic impurity function, and several stopping criteria. The latter include the minimal number of examples covered, maximal clause length, minimal local improvement, minimal description length, threshold for allowed error, and fixed variable depth. FORS has been success-

fully applied in several synthetic and real-world problem domains, including the tasks of modeling water oscillations in a surge tank and prediction of workpiece roughness in a steel grinding process.

When the value of the target variable Y is predicted as a constant or a linear combination of variables appearing in the clause, we can say that in fact a linear regression background knowledge predicate is applied. However, this predicate differs from other background knowledge predicates: it has a special built-in semantics. When applied, this predicate determines the values of the constant coefficients in the linear regression equations by taking into account the whole set of examples covered by a clause. This is in contrast to other predicates, which only take into account one example at a time when applied.

6.2 Qualitative regression in normal ILP

Qualitative regression is an approach to identifying regularities in numerical data by using background knowledge predicates with a special built-in semantics (meta-predicates) [46]. The predicate *order*$(A = B$ +/- $E)$ is such a predicate, similar to the 'big oh' notation normally used for denoting the order of magnitude in complexity theory. The *order* predicate states that the data set 'roughly' conforms to the equation $A = B$, allowing a +/- E deviation for positive instances. The predicate can use several equational forms, such as $Y = c$, $Y = aX + b$, $Y = aX^b$, $Y = a^X$, and $a^2 = X^2 + Y^2$, which relate a dependent variable Y to an independent variable X. The learning program PROGOL then determines the appropriate constants, as well as a bound on the variation of positive data from the equation.

Given examples of the target predicate $f(A, B)$, where $A = B - 3$ holds for positive examples, PROGOL induces the clause $f(A, B)$: $-$*order*$(A = B-3$ +/- $0.0)$ [46]. When an outlier $f(14.0, 30.0)$ is given as a positive example, PROGOL correctly decides that it is an outlier and maintains a separate clause for it. In [46], several additional illustrative examples are considered, showing that qualitative regression works nicely for polynomial equations, but has considerable problems with exponential ones.

6.3 Inductive Constraint Logic Programming

It is well known that Constraint Logic Programming (CLP) can successfully deal with numerical constraints. The idea of Inductive Constraint Logic Programming (ICLP) [56] is to benefit from the number-handling capabilities of CLP, and to use the constraint solver of CLP to do part of the search involved in inductive learning. To this end a maximally discriminant generalization problem in ILP is transformed to an equivalent constraint satisfaction problem (CSP) [56]. The solutions of the original ILP problem can be constructed from the solutions of CSP, which can be obtained by running a constraint solver on CSP.

Generalization in the context of constraints has been addressed in ILP in earlier work: learning of constrained atoms [49] and generation of constrained clauses that involve numerical constraints from examples [39]. The later approach was motivated by geometrical applications, where the goal is to automatically induce geometrical constraints from examples. For example, in avoiding the collision between an object and an obstacle, the region of safe moves of the object can be naturally described by a set of linear constraints.

The essential differences in the roles devoted to ILP and CLP in the above three approaches can be summarized as follows. In [49], the induction of constrained atoms is done by incorporating the structure of constraints into ILP. In [39], the construction of constrained clauses is performed by extending inverse resolution into CLP, whereas in [56] this is achieved by interleaving ILP and CLP.

6.4 Numerical constraints in nonmonotonic ILP

A recent implementation of the CLAUDIEN system for clausal discovery [9] includes the capability of using inequalities that compare a real-valued variable to a real number [62]. In this way, the handling of numerical constraints has been achieved in the nonmonotonic ILP framework. On the other hand, the system LAGRANGE [16, 18] for discovering differential equation models from an example behavior of a dynamic system has been developed. While its domain of application is not a typical ILP domain, LAGRANGE operates in the spirit of nonmonotonic ILP and the task it addresses has been formulated as a nonmonotonic ILP problem [13]. The basic idea of LAGRANGE was later incorporated in CLAUDIEN, thus enabling the discovery of concepts that mix logical and numerical constraints [13, 61].

In the ILP formulation of LAGRANGE, an example behavior of a dynamic system is represented by a ground fact of the predicate $behavior(T, X_1, \ldots, X_n)$, while the background knowledge consists of the predicates $deriv(T, X, DX)$, $sine(X, C)$, $cosine(X, C)$, $multiply(X, Y, Z)$, and $equation(DepVar, IndepVars, Coeffs)$. Example behaviors are specified by lists of measurements of a set of system variables, and background knowledge predicates enable the introduction of new variables as time derivatives, sines or cosines of system variables. New variables can be further introduced by multiplication. Finally, linear equations involving the original system variables and the newly introduced ones are tested for validity by using the predicate $equation(DepVar, IndepVars, Coeffs)$ which involves linear regression and has a special built-in semantics. In essence, it states that the dependent variable $DepVar$ can be expressed as a linear combination of the independent variables $IndepVars$, where the coefficients $Coeffs$ apply. A successor of LAGRANGE, named GOLDHORN [32], includes mechanisms for handling noisy data and has been applied to several real-life domains, including modeling algal growth in the Lagoon of Venice.

The ability to handle linear equations has been also incorporated into CLAUDIEN [13, 61]. It is based on using the predicate $equation(DepVar, IndepVars, Coeffs)$ and a constant tolerance level E, which allows some deviation of the values of the dependent variable from those predicted by the linear equation. This is realized by using the predicate $almost\_equal(DepVar, LinearExpression)$ [13], which is similar to the predicate $order(A = B +/- E)$ used in qualitative regression [46].

7 Summary and further work

In the course of the ESPRIT III Project 6020 Inductive Logic Programming, major advances in handling imperfect data have been achieved through the use of heuristics, techniques for handling missing values and sparse data, advanced search techniques and techniques for handling real-valued data.

Initial advances in heuristics continued the development of accuracy and informativity based heuristics, developed within the attribute-value learning framework. An

elaborate RELIEF-based search heuristic has also been developed and adapted to ILP. Other heuristics, more appropriate for ILP (since they take into account background knowledge) have been developed, based on the the Minimal Description Length (MDL) principle. Several encoding schemes have been proposed and their results evaluated in difficult learning problems.

LINUS replaces examples with unknown values with one or more examples without unknown values, by adapting techniques from attribute-value systems. In order to deal with missing values and too sparse training examples, COLA combines learning with conceptual clustering. Finally, RIBL deals with too sparse training examples by adopting an instance-based approach within a relational representation.

Advanced search techniques were developed, including stochastic search, genetic search, exhaustive beam search and bi-directional search. Two stochastic ILP algorithms, sFOIL and MILP, have been developed, as well as a genetic ILP algorithm GILP. The system PROGOL uses bi-directional search in addition to aleviating some of the limitations of GOLEM. An ILP approach which uses a novel declarative bias, a RELIEF-based search heuristic and exhaustive beam search for phrases of literals was developed and implemented in the ILP-R system.

In order to deal with (imperfect) real-valued data, the integration of numerical law discovery (regression) techniques within the normal and nonmonotonic ILP approaches has been achieved. A relational regression approach has been implemented, based on the DINUS transformation to propositional form and the use of existing regression tree learning systems. A relational regression algorithm FORS based on FOIL has also been implemented. Qualitative regression using meta-level predicates in background knowledge has also been studied, as well as Inductive Constraint Logic Programming which benefits from the number-handling capabilities of CLP and the potential of the constraint solver of CLP to do part of the search involved in inductive learning. Finally, numerical law discovery based on linear regression has also been incorporated into the nonmonotonic ILP semantics.

Research in this area of ILP is likely to continue along the following directions:

- Developing methods for handling numerical constraints in ILP systems, and in particular, further developing a first-order regression system that can deal with nondeterminate background knowledge, and developing techniques for handling numerical constraints in nonmonotonic ILP.

- Developing probabilistic and heuristic methods for ILP systems, and in particular, by developing a representation and methods to deal with probabilistic and uncertain concepts in ILP, and by developing heuristics that eliminate irrelevant literals and help overcome myopic behaviour of most current ILP systems.

- Developing techniques that induce or make use of constraints. In addition to the techniques that deal with numerical constraints, this includes the development of techniques for and the formalization of the induction of database constraints, the development of methods for identifying and describing full first-order constraints within ILP, and developing techniques for learning constraint logic programs.

- Developing techniques that use specific built-in semantics. In addition to methods for handling numerical constraints, this includes the exploitation of other specific built-in predicate definitions.

The developed methods can be applied to practical problems in the area of data mining and discovery, and database design. Special emphasis will be given to data mining and discovery applications. Namely, with data warehousing becoming more and more popular, there is an increased awareness that structural data and relations between data are essential in application areas such as molecular biology, ecology, control, intelligent software agents, and car design. For these applications, some industrial end-users have already shown strong interest.

Acknowledgements

The research summarized in this paper was in part supported by the ESPRIT III Project 6020 on Inductive Logic Programming. The authors acknowledge the financial support of the Slovenian Ministry of Science and Technology. Sašo Džeroski is currently also supported by the European Research Consortium for Informatics and Mathematics.

References

[1] I. Bratko and S. Džeroski. Engineering applications of ILP. *New Generation Computing* 13 (Special issue on Inductive Logic Programming), 313–333. Ohmsha, 1995.

[2] G. Bisson. Conceptual clustering in a first-order logic representation. In *Proc. 10th European Conference on Artificial Intelligence*, 458–462. John Wiley, 1992.

[3] B. Cestnik, I. Kononenko and I. Bratko. ASSISTANT 86: A knowledge elicitation tool for sophisticated users. In I. Bratko and N. Lavrač (eds.) *Progres in Machine Learning.* Sigma Press, 1987.

[4] B. Cestnik and I. Bratko. On estimating probabilities in tree pruning. In *Proc. 5th European Working Session on Learning*, 151–163, 1991.

[5] G. Chaitin. *Information, Randomness and Incompleteness - Papers on Algorithmic Information Theory.* World Scientific Press, Singapore, 1987.

[6] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning* 16: 203–225, 1994.

[7] D. Conklin and I.H. Witten. Complexity-based induction. *Machine Learning* 3(4): 261–283, 1989.

[8] L. De Raedt. *Interactive Theory Revision: An Inductive Logic Programming Approach.* Academic Press, London, 1992.

[9] L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proc. 13th International Joint Conference on Artificial Intelligence*, 1058–1063. Morgan Kaufmann, 1993.

[10] L. De Raedt, N. Lavrač and S. Džeroski. Multiple predicate learning. In *Proc. 13th International Joint Conference on Artificial Intelligence*, 1037–1043. Morgan Kaufmann, 1993.

[11] L. De Raedt and W. Van Laer. Inductive constraint logic. In *Proc. 5th Workshop on Algorithmic Learning Theory.* Lecture Notes in Artificial Intelligence, Springer, 1995 (in press).

[12] S. Džeroski. Handling imperfect data in inductive logic programming. In *Proc. 4th Scandinavian Conference on Artificial Intelligence, SCAI-93*, 111–125. IOS Press, Amsterdam, 1993.

[13] S. Džeroski. *Numerical constraints and learnability in inductive logic programming.* PhD Thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Slovenia, 1995.

[14] S. Džeroski, B. Cestnik and I. Petrovski. Using the m-estimate in rule induction. *Journal of Computing and Information Technology* 1(1): 37–46, 1993.

[15] S. Džeroski and N. Lavrač. Inductive learning in deductive databases. *IEEE Transactions on Knowledge and Data Engineering* 5 (6): 939–949. (Special Issue on Learning and Discovery in Knowledge-based Databases), 1994.

[16] S. Džeroski and L. Todorovski. Discovering dynamics. In *Proc. 10th International Conference on Machine Learning*, 97–103. Morgan Kaufmann, San Mateo, CA, 1993.

[17] S. Džeroski and L. Todorovski. Handling real numbers in inductive logic programming. In *Proc. 3rd Electrotechnical and Computer Science Conference*, Volume B, 143–146, Slovenian Section IEEE, Ljubljana, Slovenia, 1994.

[18] S. Džeroski and L. Todorovski. Discovering dynamics: from inductive logic programming to machine discovery. *Journal of Intelligent Information Systems*, 4: 89–108, 1995.

[19] S. Džeroski, L. De Haspe, B. Ruck, W. Walley. Classification of river water quality data using machine learning. In *Proc. International Conference on the Development and Application of Computer Techniques to Environmental Studies, ENVIROSOFT'94*.

[20] S. Džeroski, L. Todorovski, T. Urbančič. Handling real numbers in ILP: A step towards successful behavioral cloning. In *Proc. 8th European Conference on Machine Learning*, 283–286. Springer, 1995.

[21] W. Emde. Inductive learning of characteristic concept descriptions from small sets of classified examples. In *Proc. 8th European Conference on Machine Learning, ECML'94*, 103–121 Springer, 1994.

[22] W. Emde and D. Wettschereck. Relational instance-based learning. In *Proc. Fachgruppentreffen der Fachgruppe Machinelles Lernen der GI, FGML'95*. University of Dortmund, 1995.

[23] J. Kalbfleish. *Probability and Statistical Inference*, Volume 2. Springer, New York, 1979.

[24] A. Karalič. Induction of regression trees from incomplete data. MSc Thesis, University of Ljubljana, Faculty for Electrical Engineering and Computer Science, Ljubljana, Slovenia, 1991.

[25] A. Karalič. First order regression. PhD Thesis, University of Ljubljana, Faculty for Electrical Engineering and Computer Science, Ljubljana, Slovenia, 1995.

[26] A. Karalič. First-order regression: Applications in real-world domains. In *Proc. 2nd International Workshop Artificial Intelligence Techniques, AIT'95*, 177–190. Brno, 1995.

[27] K. Kira and L.A. Rendell. A practical approach to feature selection. In *Proc. 9th International Conference on Machine Learning*, 249–256. Morgan Kaufmann, San Mateo, CA, 1992.

[28] M. Kovačič. MDL heuristic in ILP revised. In *Proc. Descriptional Complexity Workshop* (at 11th International Conference on Machine Learning), New Brunswick, New Jersey, USA, 1994.

[29] M. Kovačič. Stochastic Inductive Logic Programming. PhD thesis, University of Ljubljana, Faculty of Electrical Engineering and Computer Science, Ljubljana, 1994.

[30] M. Kovačič. MILP - A stochastic approach to Inductive Logic Programming. In *Proc. 4th International Inductive Logic Programming Workshop ILP'94*, Bonn, Germany, 1994.

[31] M. Kovačič, N. Lavrač, M. Grobelnik, D. Zupanič and D. Mladenič. Stochastic search in inductive logic programming. In *Proc. 10th European Conference on Artificial Intelligence, ECAI-92*, 444–445, John Wiley & Sons, Chichester, 1992.

[32] V. Križman, S. Džeroski, and B. Kompare. Discovering dynamics from measured data. In *Working Notes of the MLnet Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*. Institute of Computer Science, Heraklion, 1995.

[33] N. Lavrač and S. Džeroski. Inductive learning of relations from noisy examples. In S. Muggleton (ed.) *Inductive Logic Programming*, 495–514. Academic Press, 1992.

[34] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood (Simon & Schuster), Ellis Horwood Series in Artificial Intelligence. UK: Chichester, 1994.

[35] N. Lavrač, S. Džeroski. Weakening the language bias in LINUS. *Journal on Experimental and Theoretical Artificial Intelligence* 6: 95–119, 1994.

[36] N. Lavrač, B. Cestnik and S. Džeroski. Use of heuristics in empirical inductive logic programming. In *Proc. 2nd International Workshop on Inductive Logic Programming.* ICOT TM-1182, Tokyo, 1992.

[37] N. Lavrač, S. Džeroski and M. Grobelnik. Learning nonrecursive definitions of relations with LINUS. *Proc. 5th European Working Session on Learning*, Springer, 265–281, 1991.

[38] N. Lavrač, S. Džeroski, V. Pirnat and V. Križman. The utility of background knowledge in learning medical diagnostic rules. *Applied Artificial Intelligence* 7: 273-293, 1993.

[39] F. Mizoguchi and H. Ohwada. Constraint-directed generalization for learning spacial relations. In *Proc. Second International Workshop on Inductive Logic Programming.* ICOT TM-1182, Tokyo, 1992.

[40] S. Muggleton. A strategy for constructing new predicates in first-order logic. In *Proc. 3rd European Working Session on Learning.* Pitman, 1988.

[41] S. Muggleton (ed.). *Inductive Logic Programming.* Academic Press, 1992.

[42] S. Muggleton. Bayesian Inductive Logic Programming. Invited Talk at 1994 International Machine Learning Conference and 1994 Workshop on Computational Learning Theory.

[43] S. Muggleton. Inverse Entailment and Progol. *New Generation Computing* 13 (Special issue on Inductive Logic Programming), 245–286. Ohmsha, 1995.

[44] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proc. 5th International Conference on Machine Learning*, 339–352. Morgan Kaufmann, 1988.

[45] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proc. 1st Conference on Algorithmic Learning Theory*, 368-381. Ohmsha, 1990.

[46] S. Muggleton and C.D. Page. Beyond first-order learning: Inductive learning with higher-order logic. OUCL Programming Research Group Technical Report PRG-TR-13-94. 1994.

[47] S. Muggleton, A. Srinivasan and M. Bain. Compression, significance and accuracy. In *Proc. 9th International Conference on Machine Learning*, 338–347. Morgan Kaufmann, 1992.

[48] T. Niblett T and I. Bratko. Learning decision rules in noisy domains. In M. Bramer (ed.) *Research and Development in Expert Systems III*, 24–25. Cambridge University Press, 1986.

[49] D. Page and A.M. Frisch. Generalization and learnability: A study of constrained atoms. In S. Muggleton (ed.). *Inductive Logic Programming*, 29–61. Academic Press, 1992.

[50] U. Pompe. sFOIL: Stochastic approach to Inductive Logic Programming. In *Proc. 2nd Slovenian Electrotechnical and Computer Science Conference*, Portorož, Slovenia, September 1993.

[51] U. Pompe and I. Kononenko. Linear space induction in first order logic with RELIEFF. In R. Kruse, R. Viertl and G. Della Riccia (eds.):*Mathematical and Statistical Methods in Artificial Intelligence.* CISM Course and Lecture Notes 363, 185–220. Springer, 1995.

[52] U. Pompe and I. Kononenko. Naive Bayesian classifier within ILP-R. In *Proc. 5th International Workshop on Inductive Logic Programming, ILP'95*, 417–436. Technical Report Katholieke Universiteit Leuven, 1995.

[53] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3): 239–266, 1990.

[54] J. Rissanen. Modeling by shortest data description. *Automatica*, 14: 465–471, 1978.

[55] C. Sammut and R. Banerji. Learning concepts by asking questions. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell (eds.) *Machine Learning: An Artificial Intelligence Approach, Volume 2*, 167–191. Morgan Kaufmann, 1986.

[56] M. Sebag and C. Rouveirol. Constraint Inductive Logic Programming. In *Proc. 5th International Workshop on Inductive Logic Programming, ILP'95*, 181–198. Technical Report Katholieke Universiteit Leuven, 1995.

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

[58] A. Srinivasan, S. Muggleton, R. King and M. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In *Proc. 4th International Workshop on Inductive Logic Programming*, Gesellschaft für Mathematik und Datenverarbeitung, 217–232. GMD, Bonn, 1994.

[59] A. Srinivasan, S. Muggleton and R. King. Comparing the use of background knowledge by two inductive logic programming systems. In *Proc. 5th International Workshop on Inductive Logic Programming, ILP'95*, 199–230. Technical Report Katholieke Universiteit Leuven, 1995.

[60] I. Stahl. Compression measures in ILP. In *Proc. 5th International Workshop on Inductive Logic Programming, ILP'95*, 281–296. Technical Report Katholieke Universiteit Leuven, 1995.

[61] W. Van Laer. Personal communication, 1994.

[62] W. Van Laer, and L. De Raedt. Discovering quantitative laws in inductive logic programming. In *Working Notes of the MLnet Workshop on Machine Discovery*. Artificial Intelligence Research Institute, Blanes, 1993.

[63] A. Varšek. Inductive Logic Programming with Genetic Algorithms. PhD Thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Ljubljana, Slovenia, 1993.

[64] A. Varšek. Genetic inductive logic programming. Technical report, ESPRIT III project 6020 deliverable D.LAI.3.6., 1995.