to appear in: Proc. 13th International Conference on Machine Learning,
L. Saitta (ed.), Morgan Kaufmann, 1996

Relational Instance-Based Learning

Werner Emde, Dietrich Wettschereck
GMD FIT
German National Research Center for Information Technology
Institute for Applied Information Technology
Research Division Artificial Intelligence
Schloß Birlinghoven, 53754 Sankt Augustin
Germany
werner.emde@gmd.de, dietrich.wettschereck@gmd.de

Abstract

A relational instance-based learning algorithm, called RIBL, is motivated and developed in this paper. We argue that instance-based methods offer solutions to the often unsatisfactory behavior of current inductive logic programming (ILP) approaches in domains with continuous attribute values and in domains with noisy attributes and/or examples. Three research issues that emerge when a propositional instance-based learner is adapted to a first-order representation are identified: (1) construction of cases from the knowledge base, (2) computation of similarity between arbitrarily complex cases, and (3) estimation of the relevance of predicates and attributes. Solutions to these issues are developed. Empirical results indicate that RIBL is able to achieve high classification accuracy in a variety of domains.

1 Introduction

The field of Inductive Logic Programming (ILP) has matured enough in recent years so that researchers in this field now tackle real world problems instead of hand-crafted toy-domains [King *et al.*, 1992; Muggleton *et al.*, 1992; Sommer *et al.*, 1994b]. One strength of ILP systems lies in the fact that a first-order representation is employed. Such a representation allows relations among entities to be expressed by the analyst as well as learned by the system. For example, the relative positioning of atoms to each other in molecules can most naturally be expressed in a relational representation. Such relations often carry substantial information, and are difficult to express, to understand and to maintain in propositional representations.

In this paper we describe work undertaken to address two of the shortcomings of current ILP systems: (1) Current ILP systems fail to handle continuous attribute[1] values adequately. FOIL [Cameron-Jones and Quinlan, 1994] and PROGOL [Muggleton, 1995] are among the very few exceptions that attempt to overcome this shortcoming. (2) Noise tolerance of ILP systems is generally restricted to the elimination of the negative effects of noisy examples, while no work in ILP has been published to date describing attempts to filter out noisy attributes. These issues have been heavily investigated in the field of propositional learning algorithms. In particular, the family of k-nearest neighbor or instance-based algorithms have been found well suited to handle continuous attribute values, noisy examples and noisy attributes. Instance-based learning algorithms (IBL) have been studied for more than four decades [Fix and Hodges, Jr., 1951; Kibler and Aha, 1987; Dasarathy, 1991; Wettschereck, 1994]). These studies have pointed out a number of strong points of IBL methods which are (a) often excellent performance, (b) the ability to cope with symbolic as well as continuous attribute and class values, and (c) robustness with respect to noise in the data or missing attribute values. This list of the strong points of IBL algorithms almost reads like a list of weak points of most ILP algorithms developed to date. Hence, we propose in this paper to utilize the potential of IBL algorithms to advance the state-of-the-art in ILP. IBL methods can be employed in ILP in two ways: (1) A propositional learner can be incorporated into an ILP system as in LINUS [Lavrac and Dzeroski, 1994]. This approach requires the translation of data from relational to propositional representations. The translation approach often generates a very large number of attributes and is only applicable in domains that allow a representation of cases using determinate clauses.[2]

[1]Throughout this paper we will use the term "attribute" to denote arguments of predicates.

[2]Cohen [1993] proposed an alternative translation approach that does not require determinacy.

(2) Alternatively, as described in this paper, one can develop an IBL algorithm that directly works with the relational representation.

In this paper we will describe a relational, instance-based algorithm which we term RIBL, and present promising results of an experimental evaluation. In Section 2 we identify the research issues that surface when developing a relational instance-based learner. In Section 3 we describe our solutions to these issues, while in Section 4 we describe some empirical results that highlight the feasibility of our approach and show that RIBL qualifies as an important addition to the set of first-order learning algorithms.

2 Issues in using IBL in ILP

A number of issues had to be resolved before we could apply our experience with propositional instance-based methods to relational methods. We believe that the following is common to all propositional IBL methods. Cases are represented by a fixed length vector of attribute values. Each position in that vector represents a certain attribute, and the similarity of cases is computed by combining the similarity of each pair of attribute values. Queries are classified in a two step process: First the set of the k most similar previously seen cases is determined and then these most similar cases vote on the class of the query.

We have identified three issues that occur when one applies the principle of propositional IBL methods to a relational representation:

- What constitutes a case or neighbor? In a relational representation we are given a set of facts and rules that are connected to each other through some common argument value. Hence, we first need to extract those facts from the knowledge base that together constitute a case.
- How can one compute the similarity between arbitrary cases? This issue goes beyond whether one wants to employ Euclidean distance or not. The cases constructed in RIBL may have only a few predicates in common. In other words, in a relational representation we must be able to compare apples and oranges in a meaningful manner (which does not exclude saying that their similarity is 0). Similarity computation between structured cases is an important issue in the case-based reasoning community [Börner, 1994; Voß et al., 1994; Tammer *et al.*, 1995].
- How can one estimate the relevance of predicates and attributes? Attribute weighting and selection is an important issue in propositional IBL [Wettschereck *et al.*, 1996], and we argue that this issue will be even more important for RIBL. The case construction procedure will not be able to construct cases based on anything but syntactic relations among predicates as long as it does not receive feedback from the classification procedure. Therefore, predicates that bear little or no relevance to the task at hand may unnecessarily be included in the case description. Furthermore, the values of some of the arguments of predicates with higher arity may also be irrelevant to the task. To further complicate the matter, predicates that are relevant for some cases may not even appear in the description of other cases.

Below, we will describe our solutions to these issues.

3 Relational Instance-Based Learning

On an abstract level, our relational instance-based learning algorithm, RIBL, combines different well-known techniques in a novel way. The program is composed of four main modules:

- A case generation module constructs descriptions of cases (i.e., a conjunction of literals) that serve as examples for learning or as test cases for classification.
- A module that computes the similarity of pairs of cases.
- A module that computes the relevance of predicates and their arguments.
- A module that implements distance-weighted k-nearest neighbor learning.

The following subsections describe these modules, but first let us describe the knowledge representation we are using.

3.1 Representation of the learning input

RIBL is implemented as an external tool of the knowledge acquisition and machine learning system MOBAL [Morik *et al.*, 1993; Sommer *et al.*, 1994a]. RIBL induces concepts from the knowledge represented in the knowledge base of MOBAL using an extended function-free Horn-clause representation that is para-consistent with negation [Wrobel, 1994]. The concepts "induced" by RIBL are stored within RIBL, whereas the results of other external (rule) learning tools are added to the knowledge base of MOBAL to enable the MOBAL system's powerful inference engine to apply these rules. In order to utilize RIBL's learning results MOBAL has to call the classification procedure of RIBL.

The learning input of RIBL consists of

- facts that state properties of objects and relations among objects in the domain,

- predicate declarations that define the arity of the predicates as well as the sorts and mode declarations for their arguments, and
- type declarations that classify the terms that appear as arguments of facts into values and names, and declare the representation of values.

While facts and predicate declarations (including sorts) are representational constructs of MOBAL, type declarations are supported by RIBL. Type definitions are used to differentiate between arguments that represent an object in a domain (e.g., a person or an error code) and attribute values (e.g., a number that specifies the age of a person or the price of a component). In addition, type declarations specify how attribute values are represented (i.e., by using integers, reals, sets of unordered or ordered atoms). Mode declarations specify which arguments of a predicate are input or output arguments. This information is used in the same manner as it is used in other ILP learners (e.g., FOIL or GOLEM [Muggleton and Feng, 1992]).

To illustrate the concepts proposed in this paper, we will employ the following example facts and type and mode declarations. The set of facts describes attributes of and relations among objects in a telecommunication security domain [Sommer *et al.*, 1994b]:

manager(u1), works-for(u1,d1),
works-for(u1,d2), well-known(u1),
manager(u2), works-for(u2,d3), sec-op(op1),
sub-component(c2,c3), sub-component(c1,c4),
covers-region(c2,de), covers-region(c3,de),
component(c3), component(c4)
sub-component(c4,c6), works-in(u3,d4)
operator(o1), ...

These facts describe properties of persons, companies, components in a network and their relation to each other. In addition, the knowledge base contains facts stating which person is/is not allowed to perform particular operations on components in the network, e.g.,

may-operate(jim,pabxd-44,configure)

The following predicate declaration defines 'may-operate' as a three place predicate that consumes arguments of the sorts 'person', 'component' and 'operation'. The '!' declares all arguments as input arguments. The predicate 'has-age' is declared as a two-place predicate with an input argument at the first argument position (the predicate will not be used to find persons of a particular age).

may-operate/3 :! $< person >$, ! $< component >$,
! $< operation >$.
has-age/2 :! $< person >$, $< age >$.

type : *person* : *name*.
type : *age* : *number*.

The type definitions declare that arguments of sort *person* have to be treated as name arguments and arguments of sort *age* are numbers.

3.2 Generation of Cases from unstructured Theories

Instance-based learners, as well as many ILP algorithms such as GOLEM [Muggleton and Feng, 1992], CLINT [De Raedt and Bruynooghe, 1992], and COLA [Emde, 1994a], require the construction of cases (or starting clauses) from ground facts that can be derived from a knowledge base. The construction of case descriptions is generally restricted by a syntactic or semantic bias (e.g., restricted by a depth parameter) for reasons of computational complexity (s. [Muggleton and de Raedt, 1994]).

The case generation of RIBL computes for each example a conjunction of literals describing the objects that are represented by the arguments of the example fact. Given an example fact (like the 'may-operate' fact above), RIBL first collects all facts from the knowledge base containing at least one of the arguments also contained in the example fact. These are the literals of depth 0. If a depth parameter greater than 0 is specified by the user, then RIBL determines the set of arguments contained in the literals of depth 0 minus those which occur in the example fact. Collecting all facts that contain at least one of these new arguments gives the literals of depth 1, and so on: Literals of depth N+1 are those facts that contain at least one argument that occurs as an argument of a literal at depth N, but not as an argument at depth $I < N$.

This process is restricted by user specified argument sorts, types, and modes. Information on argument sorts ensures that an object referred to by an argument V1 at depth N is only described by literals at depth N+1 if at least one argument of the new literal equals V1 and is sort compatible with V1. For example, a person named *Mac* would not be described by facts about a computer with the same name. Information on argument types is utilized to prevent the case generation module from describing an argument at depth N+1 that it is recognized at depth N as a value argument. Suppose the age of a person appears as an argument of a literal at depth N, then the module will not consider adding literals with predicates that have arguments of sort age at the next depth level. Such literals are assumed not to support the learning process. Finally, argument modes enable the module to construct cases only along input arguments. Output arguments are determined by the input arguments of the predicate and, therefore, offer no additional information.

For the following section we assume that a case description is represented as a set of tuples. Each tuple contains the set of literals of one depth:

$$CD := \{\langle 0, \textit{literals-of-depth-0}\rangle, \ldots, \langle D, \textit{literals-of-depth-D}\rangle\} \quad (1)$$

where D is a user specified depth parameter. The function L_D delivers the set of literals of depth $Depth$:

$$L_D(CD, Depth) := \{\textit{literals} \mid \langle Depth, \textit{literals}\rangle \in CD\} \quad (2)$$

For the following section we assume that RIBL has to determine the similarity of the following two facts:

$$\text{may-operate}(u1, c1, op1)\ \text{may-operate}(u2, c2, op1) \quad (3)$$

RIBL computed these case descriptions from the example facts:

$$\begin{aligned} CD_1 := \{&\langle 0, \{\text{manager}(u1), \text{works-for}(u1, d1), \\ &\quad \text{works-for}(u1, d2), \text{well-known}(u1), \\ &\quad \text{sec-op}(op1), \text{sub-component}(c1, c4)\}\rangle, \\ &\langle 1, \{\text{covers-region}(c2, de)\rangle\} \\ CD_2 := \{&\langle 0, \{\text{manager}(u2), \text{works-for}(u2, d3), \\ &\quad \text{sec-op}(op1), \text{sub-component}(c2, c3)\}\rangle, \\ &\langle 1, \{\text{covers-region}(c3, de)\rangle\} \end{aligned} \quad (4)$$

3.3 Similarity Computation for First Order Logic

The similarity computation for RIBL is a modified version of a measure for first-order logic representations proposed by Bisson [1992]. A measure similar to that by Bisson [1992] has previously been shown to be useful within a first-order conceptual clustering system [Emde, 1994a]. In contrast to Bisson's measure, our measure can be regarded as a natural extension of similarity measures for attribute-value representations. It is designed to be applicable to case descriptions as they are described in the previous section. Therefore, our measure can be used in systems using a first-order representation of knowledge with facts and rules.

The basic idea of the measure is as follows. Objects (e.g., $u1$ and $u2$) are described by values (e.g., their age and weight) and their relation to other objects (e.g., the companies they are working for). Their similarity depends on the similarity of their attributes' values (e.g., the similarity of their age) and the similarity of the objects related to them (e.g., the similarity of the companies they work in). The similarity of the related objects in turn depends on the attribute values of these objects and their relation to other objects and so on. If an object is related to other different objects by the same relation, RIBL tries to find the most similar related objects, e.g., if $u1$'s company maintains two network switches and $u2$'s only one, RIBL will use the similarity of the most similar switches to compute the similarity of $u1$ and $u2$. The rest of this section is a detailed description of the similarity measure employed by RIBL.

Given two (example) facts $P(A_{11}, \ldots, A_{1m})$ and $P(A_{21}, \ldots, A_{2m})$ and the corresponding case descriptions, the similarity measure 'sim-e' defines their similarity with respect to the user-specified depth parameter:

$$\text{sim-e}(P(A_{11}, \ldots, A_{1m}), P(A_{21}, \ldots, A_{2m})) := \frac{\sum_{i=1, i \in \text{Input-Args(P)}}^{m} \text{sim-a}^{t(P,i)}(A_{1i}, A_{2i}, 0, L_{A_{1i}}, L_{A_{2i}}, P_{A_{1i}}, P_{A_{2i}})}{\text{card(Input-Args(P))}} \quad (5)$$

where $card(S)$ denotes the cardinality of set S and Input-Args(P) denotes the set of positions of input arguments of P. In order to compute the similarity of a pair of input arguments $\langle A_{1i}, A_{2i}\rangle$ RIBL uses the function 'sim-a' that comes in several variants distinguished by the superscript $t(P, i)$:

$$t(P, i) := \begin{cases} n \text{ if i-th argument of P is a name arg} \\ \textit{number} \text{ if i-th argument of P is a number} \\ \textit{discrete} \text{ if i-th argument of P is discrete} \end{cases} \quad (6)$$

The list of literals that contain the arguments to be compared and the predicates present in this literal are required by 'sim-a' as input information. This information is extracted from the case descriptions:

$$\begin{aligned} L_{A_{fi}} &= \{P(a_1, \ldots, a_n) \mid P(a_1, \ldots, a_n) \in L_D(CD_f, 0), \\ &\qquad A_{fi} \in \{a1, \ldots, a_n\}\} \\ P_{A_{fi}} &= \{\langle P, pos\rangle \mid P(a_1, \ldots, a_n) \in L_{A_{fi}}, a_{pos} = A_{fi}\} \\ &\qquad f = 1, 2 \end{aligned} \quad (7)$$

$L_{A_{1i}}$ stands for the set of all literals in CD_1 at depth 0 that have A_{1i} as one of their arguments, $P_{A_{1i}}$ describes the predicate/position of the argument A_{1i} in $L_{A_{1i}}$.

For example, RIBL would evaluate the following expression to compute the similarity of the examples introduced in the previous section (Equation 3):

$$\text{sim-e}\begin{pmatrix} \text{may-operate}(u1, c1, op1), \\ \text{may-operate}(u2, c2, op1) \end{pmatrix} = \frac{\begin{pmatrix} \text{sim-a}^n(u1, u2, 0, L_{u1}, L_{u2}, P_{u1}, P_{u2}) + \\ \text{sim-a}^n(c1, c2, 0, L_{c1}, L_{c2}, P_{c1}, P_{c2}) + \\ \text{sim-a}^n(op1, op1, 0, L_{op1}, L_{op1}, P_{op1}, P_{op1}) \end{pmatrix}}{3} \quad (8)$$

with

$$\begin{aligned} L_{u1} &= \{\text{manager}(u_1), \text{works-for}(u_1, d_1), \\ &\qquad \text{well-known}(u1), \text{works-for}(u_1, d_2)\} \\ P_{u1} &= \{\langle\text{manager}, 1\rangle, \langle\text{works-for}, 1\rangle, \\ &\qquad \langle\text{well-known}, 1\rangle\} \end{aligned} \quad (9)$$

A corresponding similarity measure is defined for each distinct argument type. We use the following measure to compute the similarity of numbers:

$$\text{sim-a}^{number}(A, B, Depth, L_1, L_2, P_1, P_2) := 1 - (|A - B| / \text{Range}(sort(A))) \quad (10)$$

where Range($sort(A)$) denotes the observed range of values of the sort of A. The similarity of discrete values is computed as:

$$\text{sim-a}^{discrete}(A, B, Depth, L_1, L_2, P_1, P_2) := \begin{cases} 1 \text{ if } A == B \\ 0 \text{ otherwise} \end{cases} \tag{11}$$

The similarity of two arguments of type name is defined as follows:

$$\text{sim-a}^{n}(A, B, Depth, L_1, L_2, P_1, P_2) := \frac{1}{\max(\text{card}(P_1), \text{card}(P_2))} \times \sum_{\langle P, pos\rangle \in P_1 \cap P_2} \text{sim-ls}(A, B, Depth, P, pos, L_A, L_B) \tag{12}$$

where L_A and L_B are computed for each $\langle P, pos\rangle$ tuple as follows:

$$L_A = \{P(a_1, \ldots, a_n) | P(a_1, \ldots, a_n) \in L_1, a_{pos} = A\}$$
$$L_B = \{P(a_1, \ldots, a_n) | P(a_1, \ldots, a_n) \in L_2, a_{pos} = B\}$$

For example, to compute the similarity between $u1$ and $u2$ Ribl has to look at the literals with predicates 'manages', 'works-for', and 'well-known'. As there is no literal with predicate 'well-known' for $u2$, 'sim-ls' is only computed for the first argument of 'works-for' and the first argument of 'manager':

$$\text{sim-a}^{n}(u1, u2, 0, L_{u1}, L_{u2}, P_{u1}, P_{u2}) = \frac{\left(\begin{array}{l} \text{sim-ls}(u1, u2, 0, \text{works-for}, 1, L^1_{u1}, L^1_{u2}) + \\ \text{sim-ls}(u1, u2, 0, \text{manager}, 1, L^2_{u1}, L^2_{u2}) \end{array} \right)}{\max(3,2)} \tag{13}$$

with $L^1_{u1} = \{\text{works-for}(u_1, d_1), \text{works-for}(u_1, d_2)\}$ and $L^1_{u2} = \{\text{works-for}(u_2, d_3)\}$. The function 'sim-ls' computes the similarity between two arguments only with respect to literals with a particular predicate (e.g., 'works-for') and argument position, where the argument appears:

$$\text{sim-ls}(A, B, Depth, P, pos, L_A, L_B) := \begin{cases} \frac{1}{\text{card}(L_B)} \sum_{L_x \in L_A} \max_{L_y \in L_B} \text{sim-l}(A, B, Depth, P, pos, L_x, L_y) & \text{if card}(L_A) < \text{card}(L_B) \\ \frac{1}{\text{card}(L_A)} \sum_{L_x \in L_B} \max_{L_y \in L_A} \text{sim-l}(A, B, Depth, P, pos, L_y, L_x) & \text{otherwise} \end{cases} \tag{14}$$

The sum in Equation 14 is divided by the cardinality of the larger set of literals to achieve computation of perfect similarity only if the cardinality of both sets is equal. In our example, Ribl will evaluate the similarity of $u1$ and $u2$ with with respect to 'works-for' by computing:

$$\begin{array}{l} \text{sim-l}(u1, u2, 0, \text{works-for}, 1, \text{works-for}(u1, d1), \text{works-for}(u2, d3)) \\ \text{sim-l}(u1, u2, 0, \text{works-for}, 1, \text{works-for}(u1, d2), \text{works-for}(u2, d3)) \end{array} \tag{15}$$

and use the maximum of the two 'sim-l'-values divided by two. The division by two achieves that 'sim-l' computes a smaller similarity for our example than if L^1_{u2} would contain two 'works-for' facts. Equation (14) also determines the number of matches used to compute the similarity: If one case contains n literals with predicate P and the other one contains m literals with predicate P, then Equation (14) determines the least number of literals as base for the similarity computation,i.e., MIN(n,m) matches will be taken into account.

The similarity between two single literals is computed as follows:

$$\text{sim-l}(O_1, O_2, Depth, P, pos, P(A_1, \ldots, A_l), P(B_1, \ldots, B_l)) := \frac{\text{weight}(P) * \sum_{i=1}^{l} \begin{cases} \text{SIM-A if } A_i \neq O_1 \vee B_i \neq O_2 \\ 0 \quad \text{if } A_i = O_1 \wedge B_i = O_2 \end{cases}}{l - (\text{card}(\{i \mid A_i = O_1 \wedge B_i = O_2\}))} \tag{16}$$

with

$$\text{SIM-A} = \begin{cases} \text{sim-a}^{t(P,i)}(A_i, B_i, Depth+1, L_{A_i}, L_{B_i}, P_{A_i}, P_{B_i}) \times \text{arg-weight}(P, i) & \text{if } Depth =< \text{MAX-DEPTH}, t(P,i) \neq n \\ \text{base-similarity}(A_i, B_i) & \text{otherwise} \end{cases} \tag{17}$$

and

$$\begin{array}{l} L_{A_i} = \{q(a_1, \ldots, a_n) \mid q(a_1, \ldots, a_n) \in L_D(CD_1, Depth+1), A_i \in \{a1, \ldots, a_n\}\} \\ L_{B_i} = \{q(b_1, \ldots, b_n) \mid q(b_1, \ldots, b_n) \in L_D(CD_2, Depth+1), B_i \in \{a1, \ldots, a_n\}\} \\ P_{A_i} = \{\langle P, pos\rangle \mid L \in L_{A_i}, L = P(a_1, \ldots, a_n), a_{pos} = A_i\} \\ P_{B_i} = \{\langle P, pos\rangle \mid L \in L_{B_i}, L = P(b_1, \ldots, b_n), b_{pos} = B_i\} \end{array} \tag{18}$$

As long as the user-specified depth parameter is not exceeded, 'sim-l' will recursively call the function 'sim-a'. If the actual depth exceeds the maximum depth, the similarity of name arguments is computed by the function 'base-similarity', that simply counts how often both arguments appear together at the same argument positions of the same predicate and divides the sum by the maximum of the number of occurrences of the arguments [Bisson, 1992].

The main differences of Ribl's similarity measure to Bisson's measure are the following:

- The new measure takes into account that the weights of predicates and argument positions depends on the depth of their use.
- Ribl uses hierarchically organized case representations. The similarity measure of Bisson takes as input a "net of facts" about a part of a domain, such that the similarity between a pair of arguments **a-b** may depend on the similarity of

another pair of facts **c-d**. The similarity of **c-d** may in turn be computed from the similarity of **a-b** [Emde, 1994b].

- The similarity measure of RIBL can be regarded as a generalization of similarity measures employed in attribute-value instance-based learners. This is not true for Bisson's measure, due to a difference in the formula to compute the similarity of literals (17).

3.4 The k-Nearest Neighbor Classification Module

RIBL is implemented as a generalization of the propositional, distance-weighted k-nearest neighbor algorithm [Macleod *et al.*, 1987; Kibler and Aha, 1987] generalized to a relational representation. RIBL stores all training cases that are generated by the case generation tool in its knowledge base. It is well known for propositional nearest neighbor algorithms that classification of queries through a vote of several of its nearest neighbors may, in some domains, lead to superior generalization accuracy [Wettschereck, 1994]. We suspect that this is also true for relational IBL algorithms. Hence, we employ the method of leave-one-out cross-validation [Weiss and Kulikowski, 1991] to estimate the optimal number of neighbors (k) that vote on the class of a query. RIBL is "distance-weighted" since the votes of neighbors further away from the query are weighted less than the votes of nearby neighbors (i.e., Equation 19). During classification, the k nearest neighbors of each query vote on the class of the query. When asked to classify an instance q, RIBL computes its similarity to each case x in the training set (Equation 19). The k most similar neighbors are then stored in the set K and vote on the class of q:

$$p(q, c_j, K) = \frac{\sum_{x \in K} x_{c_j} * \text{sim-e}(x, q)}{\sum_{x \in K} \text{sim-e}(x, q)} \qquad (19)$$

where x is one of the k-nearest neighbors of q, and x_{c_j} is 1 if x is a member of class j. For classification tasks the class j with the largest $p(y, c_j, K)$ is output. If there is a tie among the maximal $p(q, c_j, K)$, then one of them is randomly selected.

3.5 The Predicate and Attribute Weight Estimation Module

The third research issue raised in Section 2 identified the need to assign weights to predicates as well as to their arguments to achieve best classification results. In a review of feature weighting methods, Wettschereck et al. [1996] identified RELIEFF [Kira and Rendell, 1992; Kononenko, 1994] as one of the most versatile feature weighting methods of those reviewed. The advantages of RELIEFF are that it is an iterative algorithm that converges rather quickly, and that it requires only the similarity between the objects to be weighted as input (i.e., the attributes and predicates in the case of RIBL).

A detailed description of RELIEFF was given by Kononenko [1994]. The core idea behind RELIEFF is to incrementally adjust feature weights for a given set of training examples. For each training example, the k-nearest neighbors for each output class are found. Feature weights are then decreased by a fraction of the average feature distance to the neighbors of the same class and increased by a fraction of the average feature distance to the neighbors of the other classes, where the adjustment for the other classes is weighted by the normalized class probability of each class. Naturally, it was necessary to modify the algorithm in several ways to enable it to estimate predicate weights. A separate weight was computed for each predicate and for all output arguments of all predicates at each "case-generation-depth-level" (see Section 3.2). A simple example will illustrate why the same predicate may be of varying relevance at different levels. Suppose we are trying to learn the *has-grandson* relation. In that case it is enough to know whether a person has a child who in turn has a male child. Hence, the gender of the "first-level" child is irrelevant, while the gender of the "second-level" child is highly relevant.

The most important difference of this procedure to the original RELIEFF procedure is that three different cases are distinguished when the weights of predicates are adjusted. The first case occurs when a predicate is used in the description of neighbors belonging to the same class as the current example as well as to the description of neighbors belonging to classes different from the current example's class. This corresponds to the standard case of the original algorithm. The other two cases account for the fact that no similarity value at all is computed for a predicate that does not occur in any of the k nearest neighbors of the same class as the current example, or in any of the neighbors of the other classes. In these two cases, we adjust the weights as if the similarity computed for the predicate were 0 for all neighbors irregardless of class. These two cases are most likely to occur for predicates of arity 1 such as *male(X)* or *big(Y)*. One additional modification to RELIEFF was necessary: A predicate may occur more than once in a case at the same level. In that case, the similarity values computed for each occurrence of the predicate are averaged before they are used.

4 Empirical Evaluation

Due to the fact that the similarity measure of RIBL is a generalization of the propositional case (see Section 3.3) we were able to apply RIBL to several of propositional domains. The advantage of this approach is that we were able to compare the classification results obtained by Ribl to those of our propositional implementation of kNN [Wettschereck, 1994]. We will not report specific results here, since the accuracies obtained by Ribl are comparable to those of the propositional kNN, which are reported elsewhere [Wettschereck and Dietterich, 1995]. However, these results establish that Ribl is indeed a generalization of the propositional case.

Ribl was further tested in one relational domain. The aim of this experiment was to establish whether the case generation procedure and similarity measure of Ribl were indeed correct. We chose to conduct these experiments with the relational mesh domain of Bojan Dolsak (available from the ml-archive@gmd.de, http://www.gmd.de/ml-archive). The aim of this empirical study was to investigate the feasibility of relational IBL, and to evaluate the quality of the results obtained with Ribl. Hence, we compared Ribl to the well known learning program Foil-6.2 [Quinlan, 1990; Cameron-Jones and Quinlan, 1994], which produces most discriminant generalizations. We manually selected the parameter settings for Foil-6.2 that gave the best results in these relational test domains.

Ten different Finite Element (FE) mesh models were used as a source of examples. A description of the domain can be found in [Dolsak and Muggleton, 1992]. The target predicate is `meshN(E)`, where E is a name of the edge and N is the number of finite elements on that edge. While the original data set contains descriptions of edges with up to 17 elements, we employed in our experiments only edges with 1, 2, 3, or 4 elements (318 such instances are described in the data set).

4.1 Test Method

Ten trials with randomly selected training and test instances were conducted in each domain. The classification accuracy results discussed below are averaged over these ten trials. In each trial, 30% of the instances of the goal concepts were randomly selected to test the induced concepts. From the remaining non-test instances we selected N% (see Figure 1) of each concept as training examples. Both programs (i.e., Foil-6.2 and Ribl) were supplied with exactly the same training and test sets. Foil-6.2 employed the positive examples of one class as negative examples for the other goal concepts (e.g., mesh1,..., mesh4).

The accuracy of the learning result was determined as the accuracy of the predictions made by the two learning algorithms for the test instances. In Ribl, test instances were assigned to the class that resulted in the maximal value in Equation 19. In Foil-6.2, all induced concept description were applied to each test instance. If several concept descriptions covered the same test instance, then that instance was assigned to the concept with the largest number of training examples.

4.2 Results

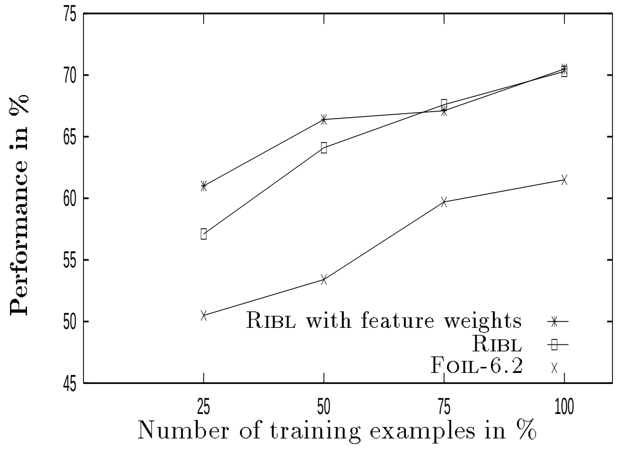

Figure 1: Learning curves for Ribl and Foil-6.2 in the Mesh domain.

The results obtained with Ribl in the Mesh domain are very encouraging (Figure 1). Ribl achieves a generalization accuracy superior to Foil-6.2's for all sizes of training sets. The performance of both algorithms improves with increasing training set sizes, as was expected. The results also indicate that predicate weights[3] improve classification accuracy in this carefully designed domain, particularly for small training sets.

5 Conclusions and Discussion

A relational instance-based learner was proposed and evaluated in a number of experiments. The results obtained from the empirical study indicate that the similarity measure employed in Ribl indeed retrieves relevant cases for a large fraction of the test cases. As elaborated in Section 3 and demonstrated in the experiments, Ribl is a generalization of the propositional k-nearest neighbor algorithm.[4] Hence, Ribl is one of the few ILP methods that function equally well with propositional and relational representations.

[3] Argument weights turned out to be less relevant in this domain.

[4] However, Ribl is computationally less efficient than its propositional predecessor, due to its more involved similarity measure and the necessity of generating cases.

It is interesting to note that even in a domain as carefully designed as Mesh, there are irrelevant predicates as determined by the predicate weight estimation procedure. This information about the relevance of certain predicates with respect to the classification task at hand can be utilized to improve either the speed or accuracy of RIBL and other inductive logic learning methods.

RIBL is a lazy learning method in that it does not construct an explicit model of the knowledge learned (as rules, for example). Hence, one advantage of many first-order learning methods, the simple description of the learned results, is lost in RIBL.

RIBL employs a specific inductive bias just as any other machine learning algorithm, and it is left to future research to describe that bias in more detail. The experiments uncovered two significant difficulties in moving IBL to a relational representation that will also be the topic of future research: (1) all test instances are classified, even if they have little similarity to any training instance, and (2) all training instances are stored, even if they are not necessary for correct classification. These shortcomings are partially addressed for propositional IBL algorithms and we will adapt some of the propositional approaches to RIBL to overcome them.

Acknowledgements

This research was partially supported by the ESPRIT basic research/long-term research projects 6020 (Inductive Logic Programming) and 20237 (Inductive Logic Programming 2). We thank David Aha, Kathy Astrahantseff, Sašo Džeroski, and Stefan Wrobel for their helpful comments on earlier versions of this papers.

References

[Bisson, 1992] G. Bisson. Learning in FOL with a similarity measure. In *AAAI92*, pages 82–87. AAAI Press, 1992.

[Börner, 1994] K. Börner. Structural similarity as guidance in case-based design. In S. Wess, K.-D. Althoff, and M. Michael M. Richter, editors, *Topics in Case-Based Reasoning: Selected Papers from the First European Workshop on Case-Based Reasoning (EWCBR-93)*, volume 837 of Lecture Notes in Artificial Intelligence, pages 197–208. Springer, 1994.

[Cameron-Jones and Quinlan, 1994] R. M. Cameron-Jones and J. R. Quinlan. Efficient top-down induction of logic programs. *SIGART Bulletin*, 5(1):33–42, 1994.

[Cohen, 1993] W. Cohen. Cryptographic limitations on learning one-clause logic programs. In *AAAI*, pages 80–85, 1993.

[Dasarathy, 1991] B. Dasarathy. *Nearest Neighbor(NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, 1991.

[De Raedt and Bruynooghe, 1992] L. De Raedt and M. Bruynooghe. Interactive concept-learning and constructive induction by analogy. *Machine Learning*, 8(2):107–150, 1992.

[Dolsak and Muggleton, 1992] B. Dolsak and S. Muggleton. The application of inductive logic programming to finite element mesh design. In S. Muggleton, editor, *Inductive Logic Programming*, pages 453–472. Academic Press, 1992.

[Emde, 1994a] W. Emde. Inductive learning of characteristic concept descriptions. In S. Wrobel, editor, *Proc. Fourth International Workshop on Inductive Logic Programming (ILP-94)*, Arbeitspapiere der GMD, 53754 Sankt Augustin, Germany, 1994. GMD.

[Emde, 1994b] W. Emde. Inductive learning of characteristic concept descriptions from small sets of classified examples. In F. Bergadano and L. De Raedt, editors, *Machine Learning: ECML-94, European Conference on Machine Learning, 1994*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 103–121, Berlin, 1994. Springer-Verlag.

[Fix and Hodges, Jr., 1951] E. Fix and J. Hodges, Jr. Discriminatory analysis, nonparametric discriminationm consistency properties. Technical Report 4, Randolph Field, TX: US Air Force, School of Aviation Medicine, 1951.

[Kibler and Aha, 1987] D. Kibler and D. Aha. Learning representative exemplar of concepts: An initial case study. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 24–30. Irvine, CA: Morgan Kaufmann, 1987.

[King *et al.*, 1992] R. King, S. Muggleton, R. Lewis, and M. Sternberg. Drug design by machine learning : the use of inductive logic programming to model the structure-activity relationship of trimethoprim analogues binding to dihydrofolate reductase. *Proc. National Academy of Sciences USA*, 89:11322–11326, 1992.

[Kira and Rendell, 1992] K. Kira and L. Rendell. A practical approach to feature selection. In *Proceedings of the Ninth International Conference on Machine Learning.* Aberdeen, Scotland: Morgan Kaufmann, 1992.

[Kononenko, 1994] I. Kononenko. Estimating attributes: Analysis and extensions of RELIEF. In F. Bergadano and L. de Raedt, editors, *7th European Conference on Machine Learning. LNAI-784*, pages 171–182. Springer Verlag, 1994.

[Lavrac and Dzeroski, 1994] N. Lavrac and S. Dzeroski. *Inductive Logic Programming – Techniques and Applications.* Ellis Horwood, New York, 1994.

[Luebbe, 1995] M. Luebbe. Datengesteuertes Lernen von syntaktischen Einschraenkungen des Hypothesenraumes fuer modellbasiertes Lernen. Technical Report LS-8 Report 15, University Dortmund, 1995.

[Macleod *et al.*, 1987] J. Macleod, A. Luk, and D. Titterington. A re-examination of the distance-weighted k-nearest-neighbor classification rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(4):689–696, 1987.

[Morik *et al.*, 1993] K. Morik, S. Wrobel, J.-U. Kietz, and W. Emde. *Knowledge Acquisition and Machine Learning: Theory Methods and Applications.* Academic Press, London, New York, 1993.

[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 and Feng, 1992] S. Muggleton and C. Feng. Efficient induction of logic programs. In S. Muggleton, editor, *Inductive Logic Programming.* Academic Press, 1992.

[Muggleton *et al.*, 1992] S. Muggleton, R. King, and M. Sternberg. Protein secondary structure prediction using logic. *Protein Engineering*, pages 647–657, 1992.

[Muggleton, 1995] S. Muggleton. Inverse entailment and progol. *New Generation Computing (Special Issue on Inductive Logic Programming)*, 13:245–286, 1995.

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

[Sommer *et al.*, 1994a] E. Sommer, W. Emde, J.-U. Kietz, and S. Wrobel. Mobal 4.1 user guide. Technical report, GMD, St.Augustin, 1996. Available via WWW at http://nathan.gmd.de/projects/ml/home.html.

[Sommer *et al.*, 1994b] E. Sommer, K. Morik, J.-M. Andre, and M. Uszynski. What online machine learning can do for knowledge acquisition — a case study. *Knowledge Acquisition*, 6:435–460, 1994.

[Tammer *et al.*, 1995] E.-C. Tammer, K. Steinhöfel, S. Schönherr, and D. Matuschek. Anwendungen des Konzeptes der strukturellen Ähnlichkeit zum Fallvergleich mittels Term- und Graph-Repräsentation. FABEL Report 38, HTWK Leipzig, Fachbereich Informatik, Mathematik und Naturwissenschaften, 1995.

[Voß et al., 1994] A. Voß et al. Retrieval of similar layouts - about a very hybrid approach in fabel. In J. Gero and F. Sudweeks, editors, *AI in Design'94*, pages 625–640, Dordrecht, 1994. Kluwer Academic Publishers.

[Weiss and Kulikowski, 1991] S. Weiss and C. Kulikowski. *Computer Systems that learn.* Morgan Kaufmann Publishers, INC San Mateo California, 1991.

[Wettschereck and Dietterich, 1995] D. Wettschereck and T. Dietterich. An experimental comparison of the nearest-neighbor and nearest-hyperrectangle algorithms. *Machine Learning*, 19:5–28, 1995.

[Wettschereck *et al.*, 1996] D. Wettschereck, D. Aha, and T. Mohri. A review of feature weighting algorithms. *Artificial Intelligence Review Journal*, 1996. accepted for publication.

[Wettschereck, 1994] D. Wettschereck. *A Study of Distance-Based Machine Learning Algorithms.* PhD thesis, Oregon State University, June 1994. Available via WWW at http://nathan.gmd.de/persons/dietrich.wettschereck.html.

[Wrobel, 1994] S. Wrobel. *Concept Formation and Knowledge Revision.* Kluwer Academic Publishers, Dordrecht, Netherlands, 1994.