Machine Learning, 38, 109–131, 2000.

Refining Numerical Constants in First Order Logic Theories

MARCO BOTTA botta@di.unito.it
ROBERTO PIOLA piola@di.unito.it
Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, 10149 Torino, Italy

Editor: Floriana Esposito, Ryszard Michalski, & Lorenza Saitta

Abstract. This paper proposes a method for refining numerical constants occurring in rules of a knowledge base expressed in a first order logic language. The method consists in tuning numerical parameters by performing error gradient descent. The knowledge base to be refined can be manually handcrafted or automatically acquired by a symbolic relational learner, able to deal with numerical features. The results of an experimental analysis performed on four case studies show that the refinement step can be effective in improving classification performances.

Keywords: theory refinement, first order logics, numerical terms

1. Introduction

Learning classification theories expressed in First Order Logic (FOL) is a hard task, and becomes even harder when dealing with numerical terms. Nevertheless, FOL is appealing because it allows one to face problems that cannot be reduced to propositional logics, such as learning from structured data of finite but unconstrained size, or handling recurrent structures. Moreover, even when a problem could be solved in a propositional setting, the solutions found in FOL are often more abstract and simpler. A serious problem to be faced in real world applications is represented by the presence of numerical terms, such as continuous-valued thresholds or parameters, which are difficult to estimate for an expert.

In the propositional setting, many induction algorithms belonging to the symbolic (e.g., Quinlan 1993; Breiman et al., 1984) and the connectionist paradigms (e.g., Rumelhart & McClelland, 1986; Jang, 1993; Moody & Darken, 1988) are capable of effectively learning numerical constants from data.

In FOL, the problem of learning "numerical" knowledge is receiving an increasing attention. In the framework of Inductive Logic Programming (ILP), a number of techniques have been devised to tackle the problem: transformation of relational problems into equivalent propositional ones, as it is done by LINUS (Lavrač & Džeroski, 1994); use of a priori "numerical knowledge" either in a procedural form, as in FOIL (Quinlan, 1990) and SMART+ (Botta & Giordana, 1993), or in a declarative form as in Progol (Muggleton, 1995); extension of TDIDT techniques (Quinlan, 1983) to FOL, as in TILDE (Blockeel & De Raedt, 1998) and STRUCT (Watanabe & Rendell, 1991); integration of numerical regression into ILP, as in FORS (Karalič & Bratko, 1997); use of constraint logic programming, as in STILL

(Sebag & Rouveirol, 1995). All these approaches are based on some form of on-line discretization or regression technique, but, in order to keep the computational complexity of the whole learning process manageable, they do not allow the fine tuning of numerical knowledge. Moreover, numerical constants are learned one at a time, as it happens in decision trees.

In a recent paper (Botta, Giordana, & Piola, 1997a), we presented a hybrid method for globally refining numerical constants occurring in FOL classification theories, using a connectionist learning scheme. The approach consists in transforming a flat classification theory into a network of elementary continuous-valued functions, corresponding to predicates and logical connectives, which can be refined by using an error gradient descent algorithm (Rumelhar & McClelland, 1986). Such a network, called First Order logic Neural Network (FONN), can be seen as an extension to FOL of KBANN (Towel & Shavlik, 1994) and of analogous methods, developed for propositional logic (Baroglio et al., 1996; Tresp, Hollatz, & Ahmed, 1993), based on Radial Basis Function Networks (RBFNs) (Poggio & Girosi, 1990; Moody & Darken, 1988), or on other paradigms (Fu, 1993; Mahoney & Mooney, 1994).

The introduction of a refinement step as a post-processing after the learning phase may have some advantages: first of all, simpler and more efficient discretization algorithms can be used in the learning phase, since the assignment of values to numerical constants can be optimized later in the refinement step. Secondly, a purely symbolic learning algorithm can be used to acquire the initial knowledge base, provided that numerical data are discretized. Moreover, if a knowledge base is elicited from an expert, he/she is not required to be very precise about numerical values, as they can be automatically adjusted later. A disadvantage is the greater total running time of the entire process.

In this paper we propose a new formulation of the FONN approach, realized in the system NTR (Numerical Term Refiner), which naturally fits the logic programming paradigm and allows one to deal with a class of theories that are more general than the flat theories considered in FONN (Botta, Giordana, & Piola, 1997a). The main advantage of NTR is that it preserves the classical logic semantics of the theories to be refined, without the need of translation between different formalisms, as was the case in FONN.

A major difference between NTR (or FONN) and all the above mentioned learning systems, which are able to deal with numerical data, is that the used error gradient descent algorithm performs a global optimization of the constants inside a clause, in contrast to the local optimization performed when literals are added one at a time.

NTR has been tested on four complex datasets, containing structured data with numeric attributes, in order to assess whether the method can actually improve performances of a knowledge base by refining its numerical constants.

In most case studies, the refinement procedure has been applied to a knowledge base previously induced from a dataset by a FOL learner (we used SMART+ (Botta & Giordana, 1993) and FOIL (Quinlan, 1990), but any of the previously mentioned systems could have been used). The acquired knowledge base contains rough estimates of numerical terms. In one case, an approximate knowledge base, given by a human expert, has also been refined. The results obtained in the refinement step show significant increases in performances on the test sets.

The paper is organized as follows: Section 2 introduces the knowledge representation formalism along with the philosophy of the approach. Section 3 describes the basic error gradient descent algorithm, whereas Section 4 extends the algorithm to multiple clauses and multiple target concepts. The experimental settings and results obtained are discussed in Section 5. Finally, Section 6 reports a discussion, and Section 7 concludes the paper.

2. Knowledge representation formalism

In this section we will briefly describe how the problem of refining numerical constants inside logic expressions can be reduced to the problem of tuning the parameters of corresponding continuous, numerical, and derivable functions, as required by the error gradient descent algorithm (Rumelhart & McClelland, 1986).

We assume the reader is familiar with standard logic programming terminology (Lloyd, 1987) and only recall here a few notions.

Examples processed by NTR are represented by specifying their elementary components, called *objects*, plus a set of their *properties* such as "color" or "length", and relationships, such as "relative position". This representation is close to the one adopted in object oriented databases (see (Giordana et al., 1997) for a more detailed description).

Theories consists of sets of clauses with the following format:

$$Head \leftarrow literal_1 \wedge \cdots \wedge literal_n$$

where *Head* is a positive predicate and $literal_i$ is a positive or negated predicate. Theories can be stratified (Apt, Blair, & Walker, 1988): stratification determines priority levels (strata) of a set of predicates, and guarantees that a predicate is not dependent on its negation. Clauses are *range-restricted*, i.e., variables in the head of a clause must also appear in its body. In the following, the last letters of the English alphabet (such as v,x,y,z) will denote variables, whereas numerical constants will be denoted by the first letters of the alphabet (a,b,c,d). Functors can only appear in the special predicate *Inside*, that is used to represent constraints on the values of numeric features. The syntactic form of this predicate is the following:

$$Inside(f(x_1, \ldots, x_s), [a, b]),$$

where $f(x_1, \ldots, x_s)$ is a real-valued function that computes the value of a numerical feature defined on objects $x_1, \ldots, x_s$, and $[a, b]$ is a closed interval in the real domain. Occasionally, f may simply be the value of a property. The semantics of predicate *Inside* is defined as usual:

$$Inside(f(x_1 \ldots, x_s), [a, b]) \; is \; \begin{cases} true & \text{if } f(x_1, \ldots, x_s) \in [a, b] \\ false & \text{otherwise.} \end{cases} \quad (1)$$

Predicates like "greater than" and "less than" have been translated into closed intervals with one extreme outside the range of interest of the feature.

Let us now introduce a simple example that will be used throughout this section for the sake of illustration. Suppose we want to define the concept "deluxe suite" in terms of "expensive" and "deluxe room", in turn defined over cost and size of the rooms. A possible theory encoding these concepts is the following:

$$\begin{aligned}
&\mathit{DeluxeSuite}(x) \leftarrow \mathit{Suite}(x) \wedge \mathit{Expensive}(x) \wedge \mathit{contains}(x, y) \wedge \\
&\qquad \wedge \mathit{DeluxeRoom}(y) \\
&\mathit{Expensive}(x) \leftarrow \mathit{Inside}(\mathit{rate}(x), [a, b]) \\
&\mathit{DeluxeRoom}(x) \leftarrow \mathit{ElegantlyFurnished}(x) \wedge \\
&\qquad \wedge \mathit{Inside}(\mathit{length}(x) \cdot \mathit{width}(x), [c, d])
\end{aligned} \tag{2}$$

A learning problem can be defined as follows:

- Given a theory T (clauses in (2)) and a learning set L (examples and counterexamples of concept *DeluxeSuite*)
- Determine an assignment of values for the numeric constants in T (a, b, c and d in (2)) in such a way to minimize the prediction error of T on the set L.

By referring to example (2), it is immediate to verify that the following operational definition[1] of the concept *DeluxeSuite(x)* can be deduced in a few resolution steps (Mitchell, Keller, & Kedar-Cabelli, 1986):

$$\begin{aligned}
&\mathit{DeluxeSuite}(x) \leftarrow \\
&\qquad \mathit{Suite}(x) \wedge \mathit{Inside}(\mathit{rate}(x), [a, b]) \wedge \mathit{contains}(x, y) \wedge \\
&\qquad \wedge \mathit{ElegantlyFurnished}(y) \wedge \\
&\qquad \wedge \mathit{Inside}(\mathit{length}(y) \cdot \mathit{width}(y), [c, d])
\end{aligned} \tag{3}$$

It should be pointed out that, if the theory contains recursive clauses, an unfolding process must be applied; this process, however, always terminates, because finite domains are considered. Moreover, all operational definitions must be supported by the provided data, i.e., they must be true of at least one example.

In general, a suite has several rooms; then, for every suite described in L, many substitutions for x, y potentially satisfying definition (3) may exist, depending on the values chosen for constants a, b, c and d. Specifically, all substitutions satisfying sub-formula:

$$\mathit{Suite}(x) \wedge \mathit{contains}(x, y) \wedge \mathit{ElegantlyFurnished}(y) \tag{4}$$

are potential candidates to satisfy definition (3). Constants a, b, c and d shall be chosen in such a way that, for every deluxe suite, at least one substitution exists that satisfies the numerical constraints stated by the two *Inside* predicates. At the same time, no non-deluxe suite should satisfy those constraints. Considering that features *rate* and *area* ($\mathit{length} \cdot \mathit{width}$) define a two-dimensional space, the learning problem can be described as in figure 1, where points represent substitutions for x, y, and the intervals $[a, b]$ and $[c, d]$ define a rectangle $R(a, b, c, d)$.

The semantics of *Inside*, defined by (1), has the drawback of being discontinuous and hence not derivable. In order to apply a gradient descent algorithm, we need to approximate

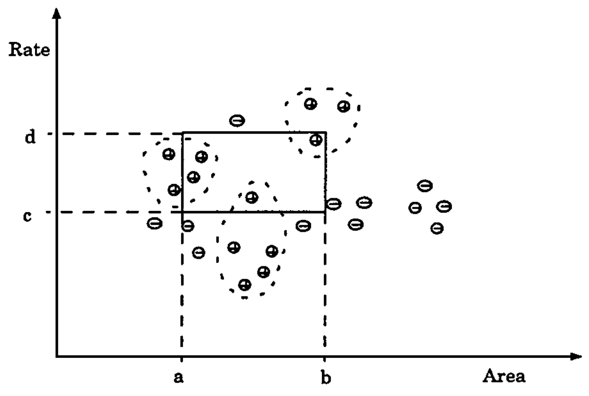


Figure 1. Representation of the learning problem in the two-dimensional space defined by the numeric values of room rate and area. The area is the product of room length and width.

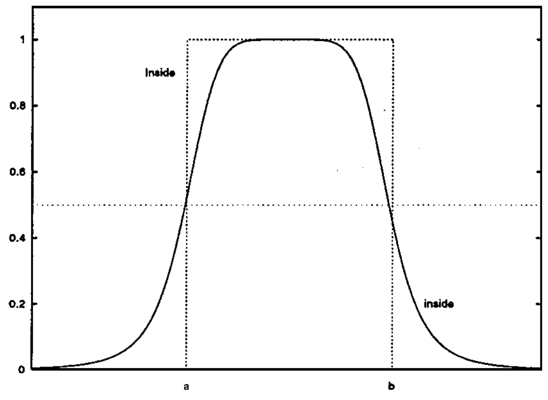


Figure 2. Approximation of a rectangular activation function *Inside*$(f(x_1, \ldots, x_s), [a, b])$ by means of a bell-shaped function *inside*$(f(x_1, \ldots, x_s), [a, b])$.

(1) with a continuous function. In this paper, we use the following bell-shaped function:

$$inside(f(x_1, \ldots x_s), [a, b]) = \frac{1}{\left(1 + \left|\left(\frac{f(x_1, \ldots x_s) - \mu_1}{\mu_2}\right)^r\right|\right)} \tag{5}$$

being $f(x_1, \ldots x_s)$ a real-valued function as before, $\mu_1 = (a + b)/2$, $\mu_2 = (b - a)/2$ and r a positive real number (see figure 2). Function (5) has been used in (Jang, 1993) for implementing continuous-valued semantics in fuzzy logic controllers. It is worth noticing that the intersection between *inside*$(f(x_1, \ldots, x_s), [a, b])$ and the straight line $y = 0.5$ corresponds to the segment $[a, b]$. Then, the conditions *inside*$(f(x_1, \ldots, x_s), [a, b]) \geq 0.5$ and *Inside*$(f(x_1, \ldots, x_s), [a, b])$ are semantically equivalent. Moreover, a conjunction of the type: $Inside(f(x_1,\ldots,x_s),[a,b]) \wedge inside(g(y_1,\ldots,y_m),[c,d])$ can be translated into:

$$\alpha(inside(f(x_1,\ldots,x_s),[a,b]), inside(g(y_1,\ldots,y_m),[c,d])) \geq 0.5 \tag{6}$$

where the symbol α denotes a combination function.

Let us consider now the logical expression:

$$\begin{aligned} &DeluxeSuite(x) \leftarrow \\ &\quad Suite(x) \wedge contains(x,y) \wedge ElegantlyFurnished(y) \\ &\quad \wedge\, \alpha(inside(rate(x),[a,b]), inside(length(y)\cdot width(y),[c,d])) \geq ø.5 \end{aligned} \tag{7}$$

Formula (7) is semantically equivalent to formula (3) only if $\alpha(f,g) \geq 0.5$ implies $f \geq 0.5$ and $g \geq 0.5$, and viceversa. We assume in the following that function α satisfies this condition.

Notice that α, in (7), can be interpreted as a measure of how close a substitution is to the center of the rectangle $R(a,b,c,d)$ in the numerical feature space. Therefore, by matching formula (7) and recording the value of α for each alternative substitution, we obtain a measure of the distance of each potential substitution from the center of rectangle $R(a,b,c,d)$. This measure can be used to guide the refinement algorithm, as described in the next section.

We still have to discuss the choice of the combination function α. In principle, function α may seem superfluous, since formula (6) can be rewritten as:

$$inside(f(x_1,\ldots,x_s),[a,b]) \geq 0.5 \wedge inside(g(y_1,\ldots,y_m),[c,d]) \geq 0.5 \tag{8}$$

However, expression (6) is more concise and allows a continuous unique activation value to be assigned to formula (7). Several combination functions for α can be used: in FONN, α was equated to the *arithmetic product*, because it is the function used in RBFNs (which FONN is inspired by) to implement logical AND, whereas α is the *min* function in NTR, which is the standard way to implement logical AND. The reason is that we want to preserve as much as possible the classical logic semantics; furthermore, expressions (6) and (8) are equivalent w.r.t. this choice, as mentioned earlier.

3. The basic learning algorithm

In this section, we describe the algorithm for refining numerical constants, starting from the case of a single unfolded clause. To this aim, the notions introduced in the previous section need to be put in a more general form. The extension to multiple clauses and multiple target concepts will be given in the next section.

Let us consider an unfolded clause in a theory T defining a concept C. It can be written in the following way:

$$C \leftarrow \varphi \wedge \psi \tag{9}$$

where φ denotes a conjunction of literals not containing any numerical constant and ψ is an assertion of type (6), containing numerical constants only. Let $\boldsymbol{\mu}$ denote the set of all such constants. Formula φ may be empty as a special case (clauses with empty ψ are not refined).

Given a set L of examples, a positive instance of C, $e^+ \in L$, must verify formula (9), whereas no negative instance $e^- \in L$ should. An instance $e \in L$ verifies a formula $\varphi \wedge \psi$ if there exists a substitution $\theta(e)$ of objects in e for the variables occurring in φ, such that numerical constraints ψ are satisfied.

For every instance $e \in L$, subformula φ in (9) may have a non empty set $\Theta(e)$ of substitutions that make it true. Then, learning consists in finding a proper assignment for the constants in ψ such that for every positive instance $e^+ \in L$, $\Theta(e^+)$ contains at least one substitution that satisfies ψ, and, for every negative instance e^- no substitution belonging to $\Theta(e^-)$ satisfies ψ. In other words, for every positive instance e^+ the function α in ψ must assume a value greater than or equal to 0.5 for at least one substitution in $\Theta(e^+)$, whereas, for every negative instance e^- the value of α must be less than 0.5 for all substitutions in $\Theta(e^-)$.

By computing the difference between 1 and the value of α for a given substitution for the positive instances, and simply taking α for the negative ones, an error measure can be defined as follows:

$$E = F_{\theta(e)}(\alpha, \tau) \tag{10}$$

where $\tau \geq 0$ is a threshold used to improve learning speed and classification accuracy by avoiding overtraining. By setting $\tau = 0$, standard error descent algorithm is performed. Two different expressions for $F_{\theta(e)}$ have been experimented, namely, the quadratic error:

$$F_{\theta(e)}(\alpha, \tau) = \begin{cases} 0 & \text{if } \alpha \geq 1 - \tau \text{ and } e \text{ denotes a positive instance} \\ (1-\alpha)^2 & \text{if } \alpha < 1 - \tau \text{ and } e \text{ denotes a positive instance} \\ 0 & \text{if } \alpha \leq \tau \text{ and } e \text{ denotes a negative instance} \\ \alpha^2 & \text{if } \alpha > \tau \text{ and } e \text{ denotes a negative instance} \end{cases} \tag{11}$$

and a thresholded cross-entropy error:

$$F_{\theta(e)}(\alpha, \tau) = \begin{cases} 0 & \text{if } \alpha \geq 1 - \tau \text{ and } e \text{ denotes a positive instance} \\ -\log(\alpha) & \text{if } \alpha < 1 - \tau \text{ and } e \text{ denotes a positive instance} \\ 0 & \text{if } \alpha \leq \tau \text{ and } e\text{denotes a negative instance} \\ \log(1-\alpha) & \text{if } \alpha > \tau \text{ and } e \text{ denotes a negative instance} \end{cases} \tag{12}$$

Error function (12) allows faster learning, but may have a negative impact on the stability of the gradient descent algorithm (Botta, Giordana, & Piola, 1997c).

Having defined an error function, the gradient descent on the error surface is performed proceeding by epochs, according to the classical off-line learning scheme, as in feed-forward neural networks (Rumelhart & McClelland, 1986). At every epoch, the whole learning set L is classified and for every parameter $\mu_k \in \boldsymbol{\mu}$, a correction is computed as the partial derivative of the error function with respect to μ_k and accumulated in a temporary variable $\Delta\mu_k$, according to the following rule:

$$\Delta\mu_k = \sum_{e \in L} -\eta \frac{\partial E}{\partial \mu_k} \tag{13}$$

being $0 < \eta < 1$ the learning rate.[2]

Afterwards, numerical constants in ψ are updated using corrections $\Delta\mu_k$.

In order to compute formula (13), two problems are still to be solved: the first one concerns the existence of derivatives of the function α, which is not continuous in NTR. The second problem concerns the selection of a particular substitution in $\Theta(e)$, which is, as well, a non-continuous operator.

The first problem can be circumvented in several ways. The one we followed consists in approximating the *min* function with a continuous one: both *arithmetic product* and *soft-min* are good candidates (see (Botta, Giordana, & Piola, 1997a) for a deeper discussion). This is a standard procedure in connectionist learning, and is justified by the fact that both the approximation and the *min* function tend to 0 when minimized and to 1 when maximized. The results presented in Section 5 have been obtained by approximating α with the *arithmetic product*, but only when computing the derivatives, and not in the classification phase.

For what concerns the selection of a substitution, let $\theta_{MAX}(e)$ be the one that produced the largest value for α. In the case there are more than one such substitutions, the first one found is chosen, without loosing generality. The gradient descent is applied to the formula instantiated with $\theta_{MAX}(e)$. The intuition behind this choice is that for a positive instance the corrections suggested by $\theta_{MAX}(e^+)$ tend to increase the value of α, whereas for a negative instance the corrections suggested by $\theta_{MAX}(e^-)$ tend to decrease the value of α. In fact, it can be proved that the direction of the gradient cannot be inverted by considering all possible substitutions, at least along the dimension corresponding to substitution $\theta_{MAX}(e)$, i.e., the derivative with respect to the parameter that receives the largest update do not change sign (Piola, 1998).

4. Extension to multiple clauses and target concepts

The basic algorithm, described in the previous section, optimizes numerical constants in the context of a single clause. Several aspects need to be discussed when generalizing this algorithm to multiple clauses and target concepts in a stratified theory.

First of all, let us consider the case of multiple clauses, but only one target concept C. This corresponds to the case of a multimodal target concept, where each clause describes one of its modality.

An extension of the learning algorithm is straightforward: clauses are unfolded and numerical constants are refined as previously described. In this way, every clause will have its own private values for the numerical constants; as a result the number of occurrences of literals increases with respect to the original theory, as well as the computational complexity of the refinement process. If closeness to the original theory is not a concern, this is an affordable solution, provided that enough instances are available for training, because the number of constants to tune may become large.

An additional difficulty, w.r.t. the single clause case is that an instance in the learning set may satisfy several clauses: in FONN, all satisfied clauses were taken into account, and their activations were combined through a perceptron function and an update value backpropagated to all the clauses. For reasons analogous to those of Section 3, only the clause with the highest activation value α is refined, in NTR. For what concerns a negative instance, the most activated clause is the one responsible for misclassification and, hence, the one that needs to be refined the most. Since the refinement phase consists of many repetitions (epochs), this strategy is consistent and converges to a minimum of the error.

If, on the contrary, the refined theory must be kept as close as possible to the original one, clause unfolding is performed as before, but numerical literals (predicates "Inside") deriving from the same clause are marked as shared. Analogously to what happens in multi-layer perceptrons, shared predicates receive and mediate updates from all the clauses that use them, so that refined clauses can be refolded to their original structure. NTR can operate by refining either independent clauses or clauses with shared predicates, depending on the way clauses are unfolded.

The extension to multiple target concepts is, again, straightforward, when concepts are independent: it is sufficient to consider one target concept at a time, to unfold its clauses, and refine them separately.

When target concepts are dependent one another (not mutually), it is always possible to define an order among them, and refine them one at a time according to the given ordering. Nonetheless, since in NTR numerical literals deriving from the same clause are shared, there is a form of hidden dependency among target concepts. As the result of the refinement process may depend upon the order of processing of the target concepts, refining a concept at a time may lead to a non optimal solution. Therefore, in NTR, as well as in FONN, multiple target concepts are learned simultaneously. To this aim, NTR uses a technique similar to that employed in multiple-output neural nets, where errors computed at every output node are backpropagated. In particular, given an instance e of a concept C_i, the best substitution $\theta_{MAX}(e)$ for each clause in the theory is considered. Then, for every target concept C_j only the most activated clause is kept. At this point, for each concept C_j an error value is computed, according to function (11) or (12), remembering that e is a positive example of C_i and a negative example of any $C_j \neq C_i$. The accumulated error is used to update the parameters $\boldsymbol{\mu}$, mediating contributions in shared predicates, as before.

In the current implementation, NTR cannot deal with (mutually) recursive target concept definitions. A possible way to face this problem, might be the adaptation of the solution used in recurrent networks (Omlin & Giles, 1996; Frasconi et al., 1996), where output signals are fed back as delayed inputs.

5. Experimental results

In order to test the effectiveness of the approach, we used four datasets containing structured instances, i.e., composed of a number of elements, each described by numerical and categorical features, whose classification depends upon both types of features. The first two are artificial datasets, purposefully designed to test the approach in a controlled environment; the other two are real-world datasets. We report results obtained by running SMART+

(Botta & Giordana, 1993) and FOIL (Quinlan, 1990) (wherever possible) to initially acquire a knowledge base, which is then refined by NTR and FONN (for comparison).

The first dataset allows us to define three typical two-class learning problems, in which either the definition for the positive or for the negative class is sufficient to perform a classification. The other three datasets are multi-class problems (10, 8 and 10 classes, respectively) and definitions for every class have to be learned. While in the first dataset the classification strategy is straightforward (instances verifying the learned definitions are considered positive), the other problems require a more sophisticated classification policy. In particular, both SMART+ and FOIL, using a crisp semantics, may classify unseen instances ambiguously, if they verify rules concluding different classes, or not classify them at all, if no rules are satisfied. The classification strategy we adopted in FONN and NTR slightly differs from the one above: since clauses have continuous activation values, we considered all clauses with an activation value $\geq$0.5, and assigned an instance to the concept defined by the clause with the greatest value. If all clauses have activation values $<$0.5, the instance is not classified.

5.1. The train check-out case study

In order to analyze the behaviour of a learning algorithm, artificial datasets offer the advantage that the learning task can be made arbitrarily complex by chosing the size of the data, the distribution of the attribute values and the rule for partitioning the learning events into positive and negative ones.

Starting from the well-known *train-going-east* problem defined by (Michalski, 1983) we extended the dataset by introducing continuous features. A car is described by a vector of four continuous and five discrete attributes: width, length, height and weight, the presence of lights and brakes, the load type (no load, passengers, normal/special materials), the type of a car (engine or not) and the number of car axles (2, 3, 4, or 6). The first car of a train is always an engine, but on long trains there is often another engine at the end, and perhaps one in the middle. An engine has a different minimum weight than standard car, and always has lights and brakes. A procedure randomly generates instances of trains, each composed by 2 to M cars, being M a parameter of the procedure.

The problem consists in deciding whether a train must not be allowed to transit on a given line (check-out procedure followed by a railway inspector), depending on the characteristics of the line (for instance, if there are bridges on the railway the train must not weigh more than the bridge can bear). Three different instances of the problem, of increasing complexity, have been created by varying the number of cars (from 1 to 3) and the kind of attributes (numerical and discrete) involved in the decision rule.

The rule used in `Task 1` is based on the value of a numerical attribute of a single car:

> A train is not allowed to go if it contains at least one car whose weight, or height, or length or width exceeds a given threshold

This task is simple because it can be easily translated into a propositional setting. However, the ability to handle structured data simplifies the learned knowledge base, by reducing the number of rules.

In `Task 2` trains are classified using rules based on numerical attributes of two or three adjacent cars:

> A train is not allowed to go if it contains two cars that are near each other and whose weights exceed a threshold w_1 or if it contains three cars that are near one another, whose weights are in an interval $[w_2, w_3]$

Two cars are considered near each other if they are at most three positions apart. This task is more complex than the previous one, because correct classification depends upon two or three cars. The first order theory learned is as compact as in the previous task, whereas a propositional counterpart turned out to be much more complicated (Botta, Giordana, & Piola, 1997b).

In `Task 3` trains are classified by a rule based on both numerical and discrete attributes of two cars:

> A train is not allowed to go if it contains two near cars, both without brakes and heavier than a threshold w_4 or if it contains two near cars carrying an unstable load (special material) and heavier than a threshold $w_5 < w_4$

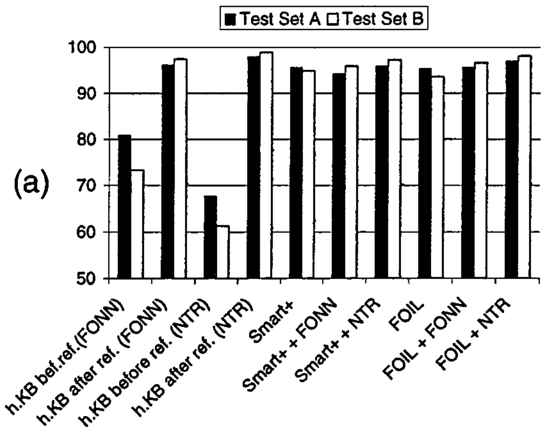
In order to test NTR's ability to generalize well over the number of seen objects (in this case, cars) that constitute a learning instance (a train), we generated three series of learning sets $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ containing 100, 200 and 500 trains, respectively, about 50% positive, whose lengths were randomly chosen between 2 and 8 cars. Two test sets have then been generated: test set $\mathcal{A}$ contains 10000 trains, whose number of cars was randomly chosen between 2 and 8, as for the learning sets; test set $\mathcal{B}$ also contains 10000 trains, but with a number of cars up to 15[3].

We ran SMART+ and FOIL on the three problems, averaging the results on five different learning sets, generated from different random seeds, and refined the theories so acquired for 1000 epochs by using FONN and NTR. Moreover, we refined three "handcrafted knowledge bases" (one per task) containing structurally correct rules, but with all the numerical terms perturbed by $\sim$30% with respect to their correct values.

Figures 3 through 5 show the percentages of correct classification, before and after the refinement step. On Tasks 1 and 2, since results obtained learning from 200 and 500 instances were quite similar, we only reported those obtained from 500 instances.

As expected, the refinement step is quite effective on the handcrafted knowledge bases, as it helps adjusting the wrong numerical terms in the theories, reaching on all tasks more than 98% correct classification. It should be pointed out that NTR is slightly better than FONN on most tasks, even though FONN combines continuous evaluation functions and some form of evidential reasoning (particularly, weighted rules and sigmoidal output activation functions). This might be due to the fact that NTR converges faster than FONN to a local minimum of the error. This is particularly evident in Task 1, where performances of NTR on the handcrafted knowledge base before refinement are significantly worse than FONN's, but after the same number of epochs performances are almost the same.

On Task 1, both SMART+ and FOIL learn quite accurate knowledge bases that cannot be improved any further by the refinement step. On Tasks 2 and 3, instead, theories learned by SMART+ are significantly improved by both FONN and NTR, whereas theories learned by



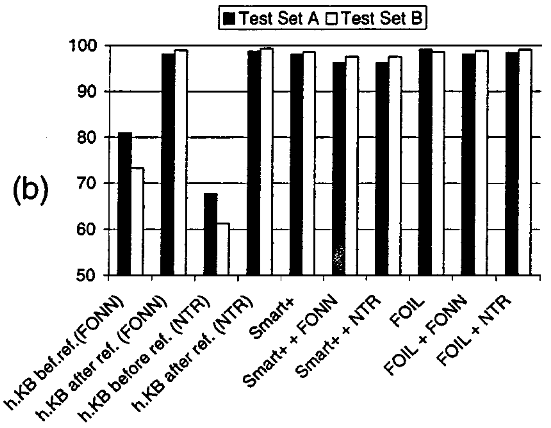


Figure 3. Recognition rate on Task 1. The graphs show the recognition rate of various knowledge bases before and after refinement. In particular, h.KB denotes a handcrafted knowledge base. This knowledge base has been applied using the classifiers embedded in FONN and in NTR, respectively, which explains the difference between the first and third columns in (a) and (b). (a) Knowledge base learned from $\mathcal{L}_1$ (100 instances). (b) Knowledge base learned from $\mathcal{L}_3$ (500 instances).

FOIL conatin many erroneous constraint literals and are only slightly improved by FONN (in this case, NTR consistently overfits the data decreasing performances). Furthermore, performances increase with the number of instances in the learning set, as expected: this is particularly evident for SMART+ in Task 3.

Another important observation concerns the ability to generalize well over the number of objects in an instance: in 36 out of 54 cases, performances after refinement improve moving from test set $\mathcal{A}$ to test set $\mathcal{B}$, and in 10 out of 54 cases performances are the same on both test sets.

Table 1 reports the execution times on Task 3: the computation time needed for refining the rules produced by SMART+ is always a small fraction of the total learning time. Moreover, NTR is faster than FONN, as it requires fewer floating point operations.

Table 1. Execution times of a single run on Task 3, for various sytems (CPU time on a SparcStation 20).

System	Learning set size	Knowledge base refined	CPU time [seconds]
SMART+	100		2140
FOIL	100		47.5
FONN (1000 epochs)	100	SMART+ (4 rules)	832
FONN (1000 epochs)	100	FOIL (5 rules)	526
FONN (1000 epochs)	100	Handcrafted (2 rules)	66.5
NTR (1000 epochs)	100	SMART+ (4 rules)	309.2
NTR (1000 epochs)	100	FOIL (5 rules)	304.2
NTR (1000 epochs)	100	Handcrafted (2 rules)	37.0
SMART+	200		8282
FOIL	200		53.8
FONN (1000 epochs)	200	SMART+ (3 rules)	1456
FONN (1000 epochs)	200	FOIL (7 rules)	856
FONN (1000 epochs)	200	Handcrafted (2 rules)	132.6
NTR (1000 epochs)	200	SMART+ (3 rules)	881.2
NTR (1000 epochs)	200	FOIL (7 rules)	444.4
NTR (1000 epochs)	200	Handcrafted (2 rules)	72.9
SMART+	500		8514
FOIL	500		354
FONN (1000 epochs)	500	SMART+ (3 rules)	879
FONN (1000 epochs)	500	FOIL (14 rules)	4205
FONN (1000 epochs)	500	Handcrafted (2 rules)	381
NTR (1000 epochs)	500	SMART+ (3 rules)	387.6
NTR (1000 epochs)	500	FOIL (14 rules)	1632
NTR (1000 epochs)	500	Handcrafted (2 rules)	181.9

As expected, FOIL (a C program) is much faster than SMART+ (a LISP program), which also uses a more sophisticated search strategy and learns more compact knowledge bases. It should be noted that the computation time of the refinement step strongly depends on the complexity of the knowledge bases (number of variables, number of rules), rather than on the number of training instances.

Finally, we ran on Task 3 SMART+ provided with a partial domain theory describing the correct constraints, but leaving the system free of learning *Inside* predicates as nedeed, and then refined for 1000 epochs the theories so acquired.

As shown in figure 6, the use of background knowledge in the learning phase helps building more correct theories, which can be further improved by the refinement step.

A conclusion from these experiments is that the proposed approach is particularly effective when knowledge bases do not need a large amount of structural refinement; nonetheless, even when the knowledge to be refined contains many erroneous constraints, performances can still be improved a little.

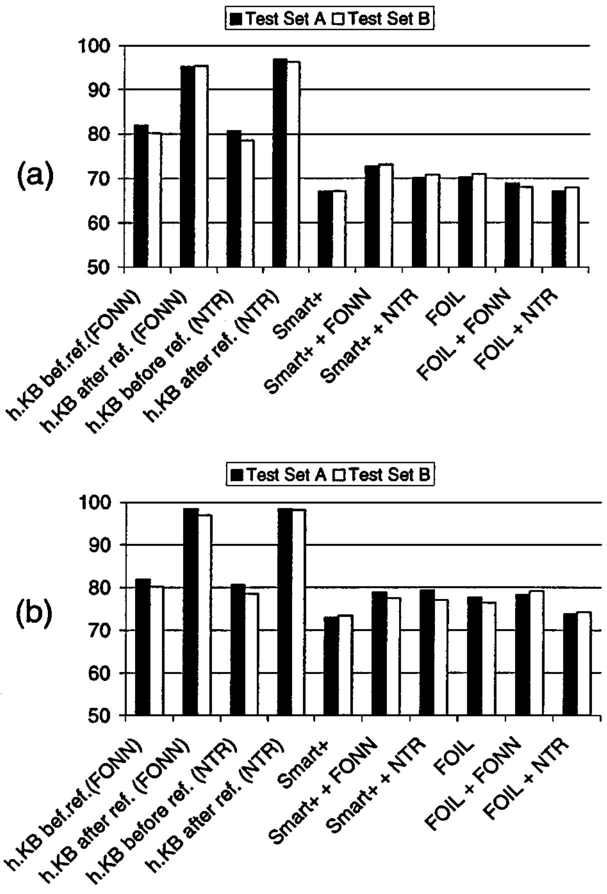
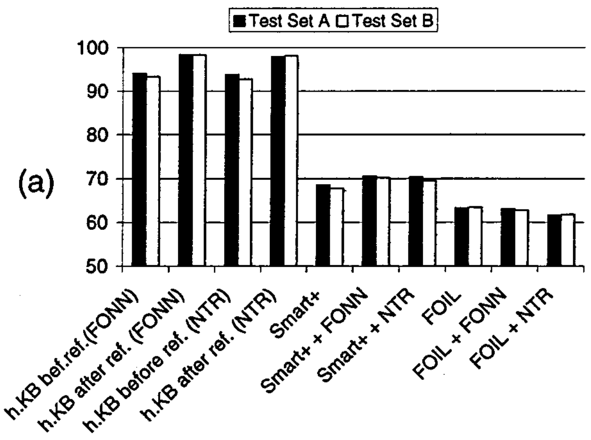


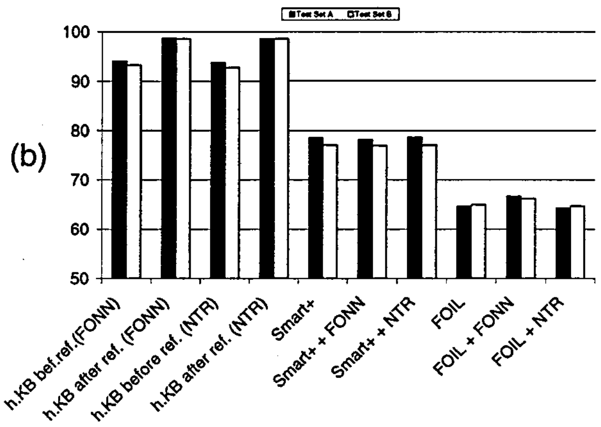
Figure 4. Recognition rate on Task 2. The notation is the same as in figure 3. (a) Knowledge base learned from $\mathcal{L}_1$ (100 instances). (b) Knowledge base learned from $\mathcal{L}_3$ (500 instances).

The main reason for this impasse is due to the symbolic nature of current first order learning systems that, being unable to properly deal with numerical data, are misled by irrelevant categorical features. As can be noted from figure 7, rules found by SMART+ and FOIL are structurally incorrect, as they contain erroneous constraints, and sometimes also contain wrong numerical terms, when compared with the ones provided by the expert. The refinement process in this case can only improve to a limited extent these knowledge bases.

5.2. *Character recognition*

The second learning problem concerns an artificial dataset that bears many resemblances to a real-world one: ten capital letters of the English alphabet have been chosen and described in terms of segments, as though acquired from a tablet. Each segment is described by the





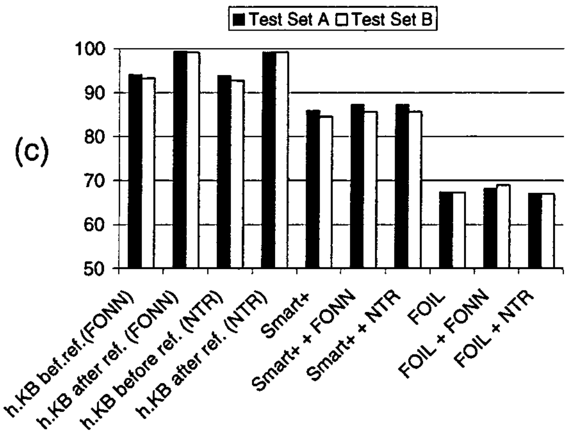


Figure 5. Recognition rate on Task 3. The notation is the same as in figure 3. (a) Knowledge base learned from $\mathcal{L}_1$ (100 instances). (b) Knowledge base learned from $\mathcal{L}_2$ (200 instances). (c) Knowledge base learned from $\mathcal{L}_3$ (500 instances).

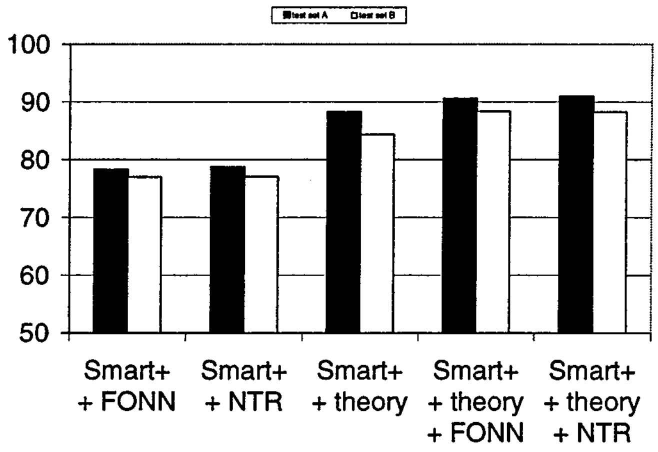


Figure 6. The effects of adding some background knowledge to the symbolic learner SMART+: performances on Task 3 of a knowledge base acquired from 200 instances. Results are averaged over five runs.

Target definition:

$$\begin{aligned}
\exists x_1, x_2 \in X.(\ & near(x_1, x_2, 1) \wedge diff(x_1, x_2) \wedge \neg brakes(x_1) \wedge \neg brakes(x_2) \wedge \\
& Inside(weight(x_1), [40; 70]) \wedge Inside(weight(x_2), [40; 70])\ ) \vee \\
\exists x_1, x_2 \in X.(\ & near(x_1, x_2, 1) \wedge diff(x_1, x_2) \wedge \neg brakes(x_1) \wedge \neg brakes(x_2) \wedge \\
& load(x_1, special-material) \wedge load(x_2, special-material) \wedge \\
& Inside(weight(x_1), [30; 70]) \wedge Inside(weight(x_2), [30; 70])\ )
\end{aligned}$$

Definition learned by SMART+:

$$\begin{aligned}
\exists x_1, x_2, x_3 \in X.(\ & diff(x_1, x_2, x_3) \wedge \neg brakes(x_1) \wedge load(x_2, special-material) \wedge \\
& Inside(weight(x_1), [43; 57]) \wedge Inside(weight(x_2), [40; 60]) \wedge \\
& Inside(width(x_3), [1.705; 2.295])\ ) \vee \\
\exists x_1, x_2, x_3 \in X.(\ & diff(x_1, x_2, x_3) \wedge \neg brakes(x_1) \wedge \neg brakes(x_2) \wedge load(x_2, special-material) \wedge \\
& Inside(width(x_2), [2.115; 2.285]) \wedge \neg brakes(x_3)\ ) \vee \\
\exists x_1, x_2, x_3 \in X.(\ & diff(x_1, x_2, x_3) \wedge \neg brakes(x_2) \wedge \neg brakes(x_3) \wedge load(x_2, special-material) \wedge \\
& Inside(width(x_1), [2.08; 2.32]) \wedge Inside(length(x_2), [17; 19])\ ) \vee \\
\exists x_1, x_2, x_3 \in X.(\ & diff(x_1, x_2, x_3) \wedge \neg brakes(x_1) \wedge \neg brakes(x_3) \wedge load(x_2, special-material) \wedge \\
& Inside(weight(x_2), [52; 68]) \wedge Inside(height(x_2), [3.4; 3.8])\ )
\end{aligned}$$

Definition learned by FOIL:

$$\begin{aligned}
\exists x_1 \in X.(\ & Inside(weight(x_1), [54.1119; 54.346])\ ) \vee \\
\exists x_1 \in X.(\ & Inside(weight(x_1), [49.6432; 50.3516])\ ) \vee \\
\exists x_1 \in X.(\ & load(x_1, normal-material) \wedge Inside(weight(x_1), [-\infty; 30.4348]) \wedge \\
& Inside(length(x_1), [-\infty; 11.2553])\ ) \vee \\
\exists x_1 \in X.(\ & load(x_1, normal-material) \wedge Inside(weight(x_1), [39.5652; +\infty]) \wedge \\
& Inside(length(x_1), [12.9311; 16.5805])\ ) \vee \\
\exists x_1 \in X.(\ & Inside(weight(x_1), [47.9661; 49.2515])\ )
\end{aligned}$$

Figure 7. Examples of learned definitions for Task 3.

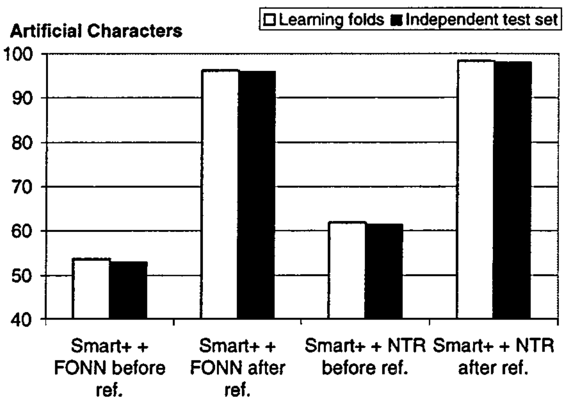


Figure 8. Performances obtained on artificial characters dataset by SMART+, FONN and NTR before and after refinement.

initial and final coordinates (x, y) in a Cartesian plane. From these basic features, other features can be extracted, such as the length of a segment, its orientation, its preceding and following segments, and so on. Some of these features are numerical by nature, whereas others are categorical. This dataset has been used to extensively test the capabilities of SMART+ (Botta & Giordana, 1993) running with several configurations. Here, we took the original dataset of 6000 instances[4] and split it into 6 folds of 1000 instances each (100 instances per class). We ran SMART+ on every fold using a default configuration (Botta & Giordana, 1993), and refined the learned knowledge from each fold for 200 epochs and tested on the remaining 5 folds, averaging the results, reported in figure 8. Moreover, we built up a new independent test set of 10000 instances (1000 per class), tested the knowledge bases and averaged the results, also reported in figure 8. The knowledge bases so acquired contain on average 98 rules, 203 literals and 118 numerical terms.

The refinement step is very effective on this learning problem, as it significantly increases performances. Moreover, moving from continuous (FONN) to boolean semantics (NTR), performances still improve a bit.

We cannot report results from FOIL, since we have not been able to configure FOIL to run on this large dataset: in some cases, it exhausted the available memory, while in other runs it found unuseful definitions, such as $A(x) \leftarrow \neg H(x)$ (x is a capitol A if it is not a capitol H).

5.3. *Document classification*

The third learning problem we addressed is a natural dataset derived from a real-world application of digitized office document classification (Esposito, Malerba, & Semeraro, 1992). In particular, single page documents containing a related collection of printed objects, such as characters, paragraphs, titles, or pictures, have been considered. An optically scanned single page document must be classified using only information about the page layout structure, i.e., the invariant geometrical characteristics shared by documents belonging to the same class, due to the underlying printing standards or writing styles. Once a document has been digitized, its page layout is produced by segmenting it through a run-length smoothing algorithm, and by grouping together some segments (or blocks) that satisfy predetermined requirements such as closeness, same type, and so on. Blocks and segments are described by categorical attributes, such as the frame type (text, line, graphics, and so on), by numerical attributes, such as width, length and positions of blocks, and relational properties, such as alignment of blocks (two blocks are aligned at left or right, bottom or top, etc.). Instances are classified into 8 classes, four corresponding to printed letters from the same company, three to magazine indexes, and the last one to a reject class representing "the rest of the world". The knowledge bases have been initially acquired by running SMART+ and FOIL on a learning set of 121 instances and refined for 500 epochs. The knowledge base learned by SMART+ contains 25 rules, 133 literals and 60 numerical terms, whereas the one learned by FOIL contains 8 rules, 37 literals and 10 numerical terms. Example of rules acquired by SMART+ are the following:

$$\begin{aligned}
&\mathit{letter}2(x) \leftarrow \mathit{document}(x) \wedge \mathit{part\text{-}of}(x, y) \wedge \mathit{north\text{-}west}(y) \wedge \\
&\quad \wedge\, \mathit{part\text{-}of}(x, z) \wedge \mathit{Inside}(\mathit{width}(z), [280, 330]) \wedge \mathit{ontop}(z, y) \\
&\mathit{spec}6(x) \leftarrow \mathit{document}(x) \wedge \mathit{part\text{-}of}(x, y) \wedge \\
&\quad \wedge\, \mathit{Inside}(\mathit{width}(y), [480, 570]) \wedge \mathit{Inside}(\mathit{height}(y), [550, 725]) \wedge \\
&\quad \wedge\, \mathit{part\text{-}of}(x, z) \wedge \neg \mathit{Inside}(\mathit{width}(z), [480, 540])
\end{aligned} \tag{14}$$

Figure 9 reports performances on a test set of 110 documents, before and after performing the refinement. From figure 9, it is evident that the refinement step is effective both in FONN and in NTR on the theory acquired by SMART+. Also on the knowledge base obtained with FOIL the refinement procedure improves performances, but starting from an incomplete theory, it can not reach the same levels of performances. Nevertheless it is surprising that, dealing with numeric attributes, the boolean semantic framework provided by NTR

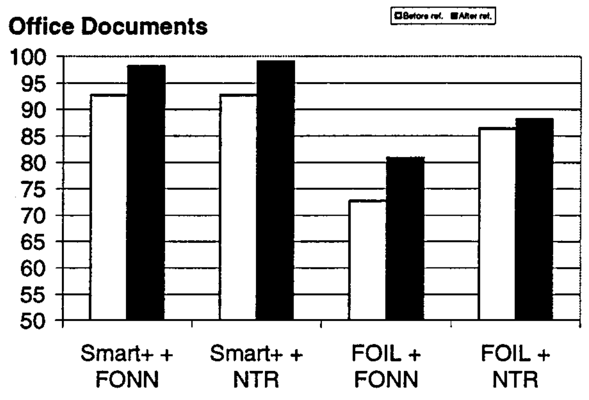


Figure 9. Performances obtained on Office Documents dataset by SMART+, FOIL, FONN and NTR before and after refinement.

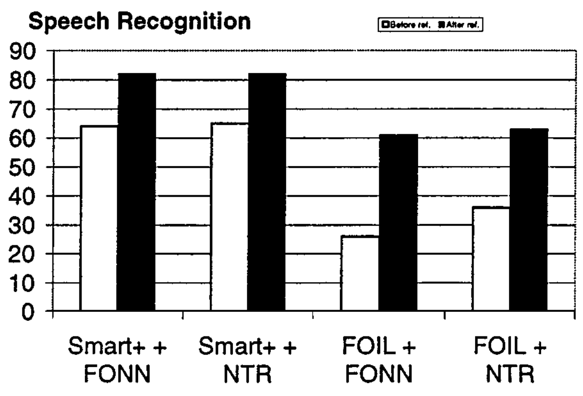


Figure 10. Performances obtained on Speech Recognition dataset by SMART+, FOIL, FONN and NTR before and after refinement.

can perform even slightly better than the flexible evidential reasoning architecture used by FONN.

Finally, we point out that the refinement procedure turned out to be very effective, because it already reached maximum performances after only 100 epochs.

5.4. Speech recognition

The fourth learning problem concerns the recognition of the ten digits spoken in Italian (Bergadano, Giordana, & Saitta, 1988) as isolated words, starting from the time evolution of two rough features, i.e., the zero-crossing and the total energy of the signal. The problem was chosen because it is sufficiently realistic to be a good test-bed, it is a really hard instance of such a kind of problems and has been previously treated by the authors. The features are extracted from the signal using classical signal processing algorithms and are then described using a set of primitives, as proposed by (DeMori et al., 1984). The theories to be refined have been learned by SMART+ and FOIL from a dataset of 219 instances.

Figure 10 reports performances on a test set of 100 instances (10 per class) before and after performing refinement for 1000 epochs. Also on this dataset the refinement step is effective, as it significantly increases performances. In this case, moving from evidential reasoning (FONN) to boolean semantics (NTR) does not increase performances. It should be pointed out, that the results reported here are much better than those obtained on this learning problem by (Bergadano, Giordana, & Saitta, 1988) (77% correct classifications), where a previous version of SMART+ was configured with all the knowledge about the domain the expert was able to provide.

6. Discussion

By globally analysing the reported experiments some lessons emerge. First of all, if the knowledge base to be refined is structurally correct, (e.g., the handcrafted knowledge bases on the trains tasks) the numerical refinement technique may provide substantial improvements, except when the initial theory is already preforming well (e.g., in Task 1 of the trains problems); in this case, in fact, the refinement step is left with little room for improvement, and performances may occasionally decrease. Finally, when the initial theory contains structurally incorrect knowledge, very modest, but statistically significant improvements in performances are observed. It is worth noting that on the multi-class learning problems, SMART+ acquires a stratified theory, in contrast to the flat set of clauses learned by FOIL, and this may partially explain the better performances obtained by these theories.

When dealing with numerical constants in first order logics, the search space becomes so huge that heuristic search strategies, such as the ones that guide the learning systems we used, are too greedy and do not allow local minima to be escaped from. The resulting knowledge bases contain erroneous constraints, and even wrong numerical literals, which cannot be fruitfully refined, neither by NTR nor by FONN. We expect that better performances can be obtained in those domains where a qualitative theory is available (both NTR and FONN were able to reach maximum performance on expert provided knowledge bases), or by combining stochastic search strategies (Burns & Danyluk, 1999) with numerical refinement. This point suggests to introduce some form of structural refinement of the theory, and provides stimuli to continue investigating the approach.

Actually, to say that NTR (and FONN) cannot change the structure of the original theory is not totally true. In fact, two kinds of structural refinement are a by-product of the refinement strategy, namely, the elimination of a numerical literal from a clause, and the elimination of a clause from the theory. The former corresponds to set numerical constants in a predicate in such a way that it is always true, whereas the latter results from setting numerical constants in such a way a predicate is always false (and so, we cannot conclude the head of the clause).

Finally, a comment about the systems chosen for the experimentation: before deciding to use FOIL for acquiring the initial knowledge bases, we also made some preliminary experiments with Progol (Muggleton, 1995) (specifically, CProgol 4.4) on the trains tasks: since the results obtained were worse than those by FOIL, as reported in Table 2, and the running time much higher than that of FOIL, we preferred to use FOIL in the experiments.

The method described is framed into the classical logic semantics setting, and special care has been devoted to the definition of a learning strategy that preserves the semantics of logical formulas during learning. By relaxing this requirement, it is possible to design slightly different learning algorithms that may produce more accurate classifiers but partially loose the original symbolic meaning. This is what happens in FONN (Botta, Giordana, & Piola, 1997a), where the outcome of the learning algorithm is a neural network that still

Table 2. Performances of Progol on the trains Tasks, averaged over five runs on a learning set containing 100 instances.

Test set	Task 1	Task 2	Task 3
$\mathcal{A}$	70.4	58.57	62.81
$\mathcal{B}$	71.2	60.15	61.1
Running time [seconds]	2396.5	1871.25	1733.21

exhibits a weak symbolic interpretability, but does not preserve the semantics of classical logic.

In the logic programming framework adopted here it is also possible, if required, to implement alternative forms of evidential reasoning, making use of certainty factors, or other numerical weights. In this case, the learning method we described can manage to learn certainty factors or rule weights, as well.

7. Conclusions

In this paper a method for learning numerical constants in classification theories, described in a restricted form of First Order Logics, has been presented. The key aspect is the translation of predicates containing numerical constants into continuous-valued functions, which can be tuned by performing the error gradient descent algorithm.

The method is based on a previous work by the same authors (Botta, Giordana, & Piola, 1997a) where an extension of Radial Basis Function Networks (Poggio & Girosi, 1990; Moody & Darken, 1988) called FONN, has been proposed in order to refine symbolic knowledge bases containing numerical information. With respect to FONN, the method presented here shows two fundamental advantages. First, it is totally embedded in the logical framework, so that it can be easily integrated with other symbolic learning strategies, whereas FONN is a totally different object that evolves independently from the original knowledge base. Second, it preserves the classical logic semantics in the formulas, whereas FONN does not.

The experimentation on several complex case studies, reported in the previous section, shows that the current implementation of the learning algorithm is able to significantly improve the classification accuracy of theories acquired by means of symbolic learners, such as SMART+ or FOIL.

Moreover, the algorithm proved to be computationally efficient and easy to tune.

Acknowledgments

We are grateful to Prof. Attilio Giordana and to Cristina Baroglio for their helpful comments.

Notes

1. By operational definition we mean that all literals in a clause body are immediately verifiable on the instances, i.e., no further resolution steps are needed to ground them, but variable substitution.
2. A momentum term has also been added in the experiments with FONN, turning (13) into $\Delta\mu_k^t = -\eta\frac{\partial E}{\partial\mu_k} + \alpha\Delta\mu_k^{t-1}$; this is a usual expedient to improve gradient descent methods in continuous networks.
3. All data are available on `http://www.di.unito.it/~mluser/datasets.html` and are fully described in (Botta, Giordana, Piola, & 1997a).
4. Available in the ML repository at UCI.

References

Apt, K., Blair, H., & Walker, A. (1988). Towards a theory of declarative knowledge. In J. Minker (Ed.) *Foundations of deductive databases and logic programming* (pp. 89–148) Los Altos, CA: Morgan Kaufmann.

Baroglio, C., Giordana, A., Kaiser, M., Nuttin, M., & Piola, R. (1996). Learning controllers for industrial robots. *Machine Learning, 23*, 221–250.

Bergadano, F., Giordana, A., & Saitta, L. (1988). Learning concepts in noisy environment. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, PAMI-10, 555–578.

Blockeel, H. & De Raedt, L. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence, 101*, 285–297.

Botta, M. & Giordana, A. (1993). SMART+: A multi-strategy learning tool. In *IJCAI-93, Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (pp. 937–943) Chambéry, France.

Botta, M., Giordana, A., & Piola, R. (1997a). FONN: Combining first order logic with connectionist learning. *Proceedings of the 14th International Conference on Machine Learning ICML-97* (pp. 48–56) Nashville, TN: Morgan Kaufmann.

Botta, M., Giordana, A., & Piola, R. (1997b). Refining first order theories with neural networks. In Z. Ras & A. Skowron. (Ed.), *Proceedings of the International Symposium on Methodologies for Intelligent Systems ISMIS-97*, LNAI (Vol. 1325, pp. 84–93) Charlotte, NC: Springer-Verlag.

Botta, M., Giordana, A., & Piola, R. (1997c). Refining numerical terms in Horn clauses. *Topics in Artificial Intelligence, proceedings of the AI*IA Conference*. LNAI (Vol. 1321, pp. 13–23) Rome, Italy: Springer-Verlag.

Breiman, L., Friedman, J., Ohlsen, R., & Stone, C. (1984). *Classification And Regression Trees*. Pacific Grove, CA: Wadsworth & Brooks.

Burns, B. & Danyluk, A. (1999). Feature selection vs theory reformulation: a study of genetic refinement of knowledge-based neural networks. *Machine Learning*, this issue.

DeMori, R., Giordana, A., Laface, P., & Saitta, L. (1984). An expert system for mapping acoustic cues into phonetic features. *Information Sciences, 33*, 115–155.

Esposito, F., Malerba, D., & Semeraro, G. (1992). Classification in noisy environments using a distance measure between structural symbolic descriptions. *IEEE Transactions on Pattern Analisys and Machine Intelligence, 14*(3), 390–402.

Frasconi, P., Gori, M., Maggini, M., & Soda, G. (1996). Representation of finite state automata in recurrent radial basis function networks. *Machine Learning, 23*, 5–32.

Fu, L. (1993). Knowledge-based connectionism for revising domain theories. *IEEE Transactions on Systems, Man and Cybernetics, 23*(1), 173–182.

Giordana, A., Neri, F., Saitta, L., & Botta, M. (1997). Integrating multiple learning strategies in first order logics. *Machine Learning, 27*, 209–240.

Jang, J. (1993). ANFIS: Adaptive-Network-Based Fuzzy Inference System. *IEEE Transactions on Systems, Men and Cybernetics, 23*(3), 665–687.

Karalič, A. & Bratko, I. (1997). First order regression. *Machine Learning, 26*, 147–176.

Lavrač, N. & Džeroski, S. (1994). *Inductive logic programming: techniques and applications*. Chichester, UK: Ellis Horwood.

Lloyd, J. W. (1987). *Foundations of logic programming*. Berlin, Germany: Springer.

Mahoney, J. & Mooney, R. (1994). Comparing methods for refining certainity-factor rule-bases. *Proc. of the Eleventh Internetional Workshop on Machine Learning ML-94*, Rutgers University, NJ.

Michalski, R. (1983). A theory and methodology of inductive learning. In R. Michalski, J. Carbonell, & T. Mitchell, (Eds), *Machine learning: an artificial intelligence approach* (pp. 83–134) Los Altos, CA: Morgan Kaufmann.

Mitchell, T., Keller, R., & Kedar-Cabelli, S. (1986). Explanation based generalization: an unifying view. *Machine Learning, 1*, 47–80.

Moody, J. & Darken, C. (1988). Learning with localized receptive fields. In T. Sejnowski, D. Touretzky, & G. Hinton (Eds.), *Connectionist models summer school*, Carnegie Mellon University.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing, 13*, 245–286.

Omlin, C. & Giles, C. (1996). Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM, 43*(6), 937–972.

Piola, R. (1998). *Refinement of knowledge bases in first order logics by means of neural networks*. Ph.D. Thesis, Università di Torino, Italy.

Poggio, T. & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE, 78*(9), 1481–1497.

Quinlan, R. (1983). Efficient classification procedures. In J. Carbonell, R. Michalski, & T. Mitchell (Eds.), *Machine learning, an artificial intelligence approach*. Morgan Kaufmann.

Quinlan, R. (1990). Learning logical definitions from relations. *Machine Learning, 5*, 239–266.

Quinlan, R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, CA.

Rumelhart, D. E. & McClelland, J. L. (1986). *Parallel distributed processing: explorations in the microstructure of cognition, Parts I & II*. Cambridge, Massachusetts: MIT Press.

Sebag, M. & Rouveirol, C. (1995). Polynomial-time learning in logic programming and constraint logic programming. *Proc. of ILP-95* (pp. 105–126) LNAI (Vol. 1297).

Towell, G. & Shavlik, J. (1994). Knowledge Based Artificial Neural Networks. *Artficial Intelligence, 70*(4), 119–166.

Tresp, V., Hollatz, J., & Ahmad, S. (1993). Network structuring and training using rule-based knowledge. In S. Hanson, J. Cowan, & C. Giles (Eds.), *Advances in neural information processing systems 5 (NIPS-5)*, (pp. 871–878) San Mateo, CA: Morgan Kaufmann.

Watanabe, L. & Rendell, L. (1991). Learning structural decision trees from examples. *Proc. of the 12th International Joint Conference on Artificial Intelligence, IJCAI-91* (pp. 770–776), Sidney, Australia.

Received December 22, 1998

Accepted June 18, 1999

Final manuscript June 18, 1999