Handling continuous-valued attributes in Incremental First-Order Rules Learning

Teresa M.A. Basile, Floriana Esposito, Nicola Di Mauro, and Stefano Ferilli

Department of Computer Science, University of Bari, Italy
{basile, esposito, ndm, ferilli}@di.uniba.it

Abstract. Machine Learning systems are often distinguished according to the kind of representation they use, which can be either propositional or first-order logic. The framework working with first-order logic as a representation language for both the learned theories and the observations is known as Inductive Logic Programming (ILP). It has been widely shown in the literature that ILP systems have limitations in dealing with large amounts of numerical information, that is however a peculiarity of most real-world application domains. In this work we present a strategy to handle such information in a relational learning incremental setting and its integration with classical symbolic approaches to theory revision. Experiments were carried out on a real-world domain and a comparison with a state-of-art system is reported.

1 Introduction

Traditional application of Machine Learning to intelligent systems development involves collecting a set of training examples, expressing them in a representation language such that the corresponding representation space facilitates learning, and using a learning algorithm to induce a set of concepts from the codified training examples. The induced concepts are subsequently validated, incorporated into an inferential system and deployed into an environment. It is well known that the choice of the proper representation for a learning problem has a significant impact on the performance of learning systems. In traditional learning problems, the training examples are typically represented as vectors of attribute-value (*attribute-value or propositional learners*). In most complex real-world domains, however, the exploitation of this kind of representation language could affect the learning efficiency. In such domains, the adoption of a more powerful representation language is required; for instance, the language of first-order logic, a natural extension to propositional representations, is able to describe any kind of relation between two or more objects. The systems that exploit such a representation are called *relational* or *Inductive Logic Programming* learning systems. Differently from the propositional representation, however, in which only a single mapping is possible between descriptions, this representation language allows a potentially large number of mappings between descriptions. Thus, it becomes more difficult to handle relationships among data and, in particular, numerical information due to the non-determinacy in relational learning, i.e. the possibility of associating more than one value to the numerical attributes for examples. This is not true for the propositional case in which the correspondence between each attribute and its value is one-to-one for each example.

The problem is accentuated if we consider the approach exploited in the learning task. Classical approaches, using *batch* methodologies, assume/require that all the information needed is available at the beginning of the learning process. This is clearly not generally true for the knowledge assimilation process, in particular in those real-world applications that require the ability of incrementally revising a domain theory as new data is encountered. In these cases *incremental* learning, as opposed to batch learning, is needed. However, the management of numerical information in incremental approaches becomes more complex due to the fact that the induction of numerical attributes must take into account the non-determinacy in the relational learning and must be performed and revised each time a new observation is encountered.

In this paper, after a brief description of the works concerned with topic of handling numerical information in relational learning, we present a strategy to handle such information in an incremental relational learning setting and its integration with classical symbolic approaches to theory revision. Some experiments were carried out on a real-world domain and a comparison with a state-of-art system is reported.

2 Related Work

Inductive Logic Programming (ILP for short) algorithms have shown their inadequacy in dealing with the large amounts of numerical information that characterize most real-world application domains. This problem has been addressed by means of different approaches. In propositional learning, discretization has received a lot of attention [6, 8, 9] and it has proven to be a valuable technique with respect to efficiency and accuracy. The general idea of discretization [16, 13] consists in splitting a range of continuous values into intervals so as to provide useful information about classes. Moving towards relational learning, the first question in ILP is: *what to discretize?*. Indeed, in relational learning each attribute in the attribute-value representation is mapped in an *argument* of a predicate. So, instead of discretizing a numerical attribute, we have to discretize a numerical argument. The problem now is that each numerical argument has a corresponding query that generates all the values of the numerical argument that can have 0,1 or more values *per example*.

A number of methods have been devised to solve the problem: The first class of methods, as in LINUS [14], concerns the transformation of relational problems into equivalent propositional ones in order to handle real numbers by means of techniques already tested in decision tree induction systems. Different methods of propositionalization have been implemented in REPART [21] and in ICP [17]. In the former the propositionalization pattern is provided by the user, while in the latter it is built from the training set. More recently, a lazy propositionalization method that selectively propositionalizes the first-order logic training set by interleaving attribute-value reformulation and algebraic resolution has been proposed [1]. The other classes of methods born for handling numerical features in an ILP setting use *a priori* knowledge, either in a procedural form as in FOIL [5] or in a declarative form as in Progol [15]. FOIL automatically produces comparative literals by means of built-in relational predicates expressed on numerical variables already present in other non-comparative literals and compared with a threshold. The semantics of the built-in relational predicates, as well as the heuristics for the selection of the best threshold, are defined procedurally and embedded in the code of the system. On the contrary, Progol explicitly *codes* the capability of numerical reasoning in a declarative form as background knowledge.

Other techniques involved the classical statistical data analysis research area that was extended with new tools of formalization in order to deal with symbolic data. This kind of integration is more complex when using first-order computational learning models due to the practical and theoretical possibility of integrating different computational strategies, different knowledge representations and different processing methods in a common framework. In [12] the authors faced the problem of handling both numerical and symbolic data in first-order models, distinguishing the phase of *model generation* from examples, concerning the on-line discretization of numerical attributes and relations, and the phase of *model recognition* by means of a flexible probabilistic coverage test. Other hybrid approaches make cooperative techniques coming from ILP and Constraint Logic Programming by means of a transformation step of counter-examples into constraints [17]. The approach proceeds generating the equivalent Constraint Satisfaction Problem (CSP) of an ILP problem: all constrained clauses that cover positive examples and reject negative ones can be trivially derived from the solutions of the CSP corresponding to the original ILP problem. Successively, the CSP is solved by means of a constraint solver and this will allow to build the set of solutions in terms of hypotheses space.

Algorithms for mapping propositional Horn clauses to neural networks have been presented by several authors [19, 2, 20]. The basic idea underlying the proposed strategies in such context is that a propositional theory can be functionally described by an `and/or` graph having atomic expressions in leaves. Such a graph can be transformed into a neural network by replacing the boolean $\wedge$ and $\vee$ operators in the nodes with continuous, derivable functions and by adding weights on the links. This method was extended and adapted to first-order theories, First Order Neural Network (FONN), as in [4], where it is implemented a method to translate a first-order classification theory into a neural network of elementary computational units that are refined, by means of an error gradient descent, preserving the readability of the theory. The method works on a theory that is supposed to be already generated by a relational learner or manually given by a domain expert, and tries to refine exclusively the numerical information expressed on the arguments of the predicates. Finally, some works to face the problem of managing numerical data concern the combination of ILP strategies with Evolutionary Computation techniques. Examples of this kind of approaches are SMART+ [3] and the work by Divina et al. [7]. In the former case the system transforms the real numbers into discrete integers that can be encoded as binary numbers, and exploits genetic algorithms to handle such values. In the latter case, the Authors propose to handle numerical attributes by using the relational built-in constraints. During the execution of a genetic-algorithm-based inductive concept learner, a constraint for a numerical attribute is generated when that attribute is selected and, successively, it is modified during the evolutionary process by using genetic operators, that were defined by the Authors and that exploit information on the distribution of the values of attributes in order to update the interval boundaries of the constraints.

All the methods above mentioned are based on the assumption that batch approaches are applied to the learning task. Here we address the problem of handling numerical information in an incremental setting that exploits first-order logic as a representation language.

3 Handling Continuous-valued Data in an incremental FOL learning system

In this section we present the extensions of the incremental ILP learning system INTHELEX [11] to make it able to handle numerical data. In such a system two entities are involved in the learning process, observations and hypotheses. Both are expressed by means of Datalog clauses (function-free Horn clauses - only constants or variables are allowed in the description), and are interpreted under the assumption of the Object Identity (OI) (*terms denoted in different way have to represent distinct objects in the description* [18]).

In the pure symbolic representation, both observations and hypotheses are represented using the same formalism, and differ in the kind of arguments on which their literals are built: constants in case of the observations, variables in case of the hypotheses. Now we need to extend the representation formalism to introduce numerical values in both observations and hypotheses. Trivially, in the observations we introduce the numerical constants and represent them in the same way of the symbolic ones; in the hypotheses, each numerical value is associated to a variable, that in the following we will call *numerical variable*, on which it is defined an interval properly determined on the grounds of the observations. Thus we can distinguish, in the observations: *numerical predicates* (iff at least one of its arguments is a numerical constant); *symbolic predicates* (iff its arguments are all symbolic constants); *numerical constants* (numeric strings); *symbolic constants* (alphanumeric strings that begin with a lower-case letter). As to the hypotheses, we firstly need to say that, in the following, a numerical constraint on a variable is so defined:

Definition 1 (Numerical Constraint). *A numerical Constraint C on a variable V is a disjunction of numerical intervals* ($Range_1$; $Range_2$; $\ldots$; $Range_n$) *where* $Range_i = (Inf, Supp)$ *for* $i = 1 \ldots n$ *and*
$Inf = (V \geq$ *numerical value*$)$ *OR* $Inf = (V > $ *numerical value*$)$
$Supp = (V \leq$ *numerical value*$)$ *OR* $Supp = (V <$ *numerical value*$)$

Thus, in the hypotheses we can distinguish: *numerical predicates* (iff at least one of its arguments is a numerical variable); *symbolic predicates* (iff its arguments are all symbolic variables); *symbolic variables* (variable - alphanumeric strings that begin with an upper-case letter); *numerical variables* (variable - alphanumeric string that begin with an upper-case letter) on which a numerical constraint is defined); *numerical constraints* reported above. It is worth noting that a numerical predicate always contains at least one symbolic term.

3.1 Modification to the Object Identity Framework

The extension of the representational language to make it able to describe numerical information in observations and hypotheses, highlighted a practical problem as regards the OI assumption that is used in the framework. Indeed, since the same numerical value could be associated as property of more than one object (e.g., same width for two blocks), we cannot impose the constraints derived from such assumption on numerical data. Thus, the transformation from a Datalog clause to its corresponding Datalog$^{\mathrm{OI}}$ (Datalog clause under the OI assumption) is reformulated in the following way:

Definition 2. *Any Datalog clause* $C = \{L_1,\ L_2, \ldots,\ L_n\}$ *has a corresponding Datalog*^{OI} *clause* $C_{OI} = \{\ core(C_{OI})\ \bigcup constraints(C_{OI})\ \}$*, where:*
$core(C_{OI}) = \{\ L_1,\ L_2, \ldots,\ L_n\ \} = C$
$constraints(C_{OI}) = \{\neq(t_1, t_2) \mid t_1,\ t_2 \in terms(C),\ t_1 \neq t_2\ \ forall\ \ terms(C) \notin numeric\_terms(C)\}$
where $numeric\_terms(C)$ *represents the set of constants that are numerical values or the set of variables that represent a numerical property.*

Furthermore, the notion of substitution, regarded as a function mapping variables to terms, that under the OI assumption should be an injective mapping, has to be revised taking into account the above considerations. Thus, given a set of terms T, we say a substitution σ is an *OI-substitution with respect to* T if and only if for all couples of terms $t_1,\ t_2 \in T$ that are not number we have: $t_1 \neq t_2 \Rightarrow t_1\sigma \neq t_2\sigma$.

Let us now see how it is possible to formally define refinement operators, starting from the definitions given in [10], in order to take into account the numerical information on which the OI constraint is not valid.

Definition 3 (Downward Refinement Operator ρ_{OI}). *Let* C *be a Datalog*^{OI} *clause. Then, a clause* D *belongs to the set of the downward refinements of the clause* C*, i.e.* $D \in \rho_{OI}(C)$*, when either of these conditions hold:*

(i) $D = C\theta$*, where* $\theta = \{X/a\}$*,* $X \in vars(C)$*,* $a \notin symbolic\_consts(C)$*, that is,* θ *is a substitution replacing a symbolic variable of* C *with a new symbolic constant, i.e. not already present in C;*
(ii) $D = C\theta$*, where* $\theta = \{X/a\}$*,* $X \in vars(C)$*,*
$a \in numeric\_consts(Numerical\_Constraint)$*, that is,* θ *is a substitution replacing a numerical variable of* C *with a numerical constant belongs to* $Numerical\_Constraint$ *that is the numerical constraint defined on the numerical variable that one is replacing;*

(iii) $D = C \cup \{\neg A\}$, *where* $\neg A$ *is a symbolic literal, such that:* $\neg A \notin C$;

(iv) $D = C \cup \{\{\neg A\}, \{Numeric\_Constraints\}\}$, *where* $\neg A$ *is a numerical literal, such that:* $\neg A \notin C$. $\{Numeric\_Constraints\} \notin C$ *is the set of numeric constraints defined on the numeric variables that are arguments of the literal* A *(note that if* A *does not contains variables, but only constants this set is empty).*

Definition 4 (Upward Refinement Operator $\delta_{\mathbf{OI}}$). *Let* C *be a Datalog*^{OI} *clause. Then, a clause* D *belongs to the set of the upward refinements of the clause* C, *i.e.* $D \in \delta_{OI}(C)$ *when either of these conditions hold:*

(i) $D = C\sigma$, *where* $\sigma = \{a/X\}$, $a \in symbolic\_consts(C)$, $X \notin vars(C)$, *that is,* σ *is an anti-substitution replacing a symbolic constant of* C *with a new variable;*

(ii) $D = C \setminus \{\neg A\}$, *where* $\neg A$ *is a symbolic literal, such that:* $\neg A \in C$;

(iii) $D = C \setminus \{\{\neg A\}, \{\neg Numeric\_Constraint\}\}$, *where* $\neg A$ *is a numerical literal, such that:* $\neg A \in C$ *and* $\{\neg Numeric\_Constraint\} \in C$ *is the set of numerical constraints that are defined on the set of numerical terms of the literal* A.

Note that: $C = H, \neg B_1, \ldots, \neg B_n \equiv (H : -B_1, \ldots, B_n)$ thus, adding/removing $\neg A$ from C is equivalent to add/remove a literal from the body clause. The operators above defined fulfill the same properties, that are deemed as desirable from a theoretical point of view, as the symbolic ones reported in [10].

3.2 Refinement Operators

Here we present a sketch of the procedure implemented by the refinement operators exploited in the system in order to handle the numerical information and their integration with the classical inductive refinement operators, that work on symbolic information, embedded in the system in consideration. When a new observation is taken into account three situations can happen: The observation (positive or negative) is correctly classified; The observation is positive and the system does not correctly classify it; The observation is negative and the system does not correctly classify it. In the first case the theory is correct and thus it does not need any revision. In the second (respectively, the third) case the theory is too specific (respectively, too general) and, thus, it needs to be revised: specifically, a generalization phase, or upward refinement, (respectively, a specialization phase or downward refinement) is required. The classical inductive refinement operators exploited in the framework operate in the following way:

- Generalization phase - Inductive Upward Refinement:
  1. eliminating one (or more) literal(s) from the clause that does not cover the positive observation;
  2. adding a new clause to the theory - the observation itself with constants properly turned into variables.
- Specialization phase - Inductive Downward Refinement:
  1. adding one (or more) literal(s), coming from the past positive observations to the clause that covers the negative observation;
  2. adding the negation of a literal, coming from the negative observation that caused the misclassification, to the clause that covers the negative observation.

A modification of these operators to handle numerical information is necessary. Firstly, we present the definition of inductive refinement operators able to handle numerical information. Then a framework for the integration of the operators that work on both numerical and symbolic information will be provided. As regards their definition, it is articulated in two phases, as reported in the following, one for the addition of a new numerical constraint and the other for the modification of an existing numerical constraint.

- **addition of a new numerical constraint.** Firstly, for each numerical constant c found in the observation it is replaced with a new variable V in the literal it is argument of. Then, to such a new variable V it is associated the proper numerical constraint based on the value of the numerical constant in the observation. In other words, given an observation, let $descr(c_1, c_2, c_3, \ldots, c_n)$ a numerical literal in such a observation. Now suppose that the **i-th** argument $(i = 1, \ldots, n)$ is a numerical constant which value is k; then it will become: $descr(c_1, c_2, \ldots, V_i, \ldots, c_n)$, $(V_i \geq k, V_i \leq k)$.

  Example 1. Let C the following clause:
  ```
  bicycle(a) ← part_of(a,b), part_of(a,c), wheel(b), large(b,10),
               wheel(c), large(c,15), distance(b,c,10).
  ```
  The resulting clause is:
  ```
  bicycle(a) ← part_of(a,b), part_of(a,c),
               wheel(b), large(b,Y), (Y >= 10, Y =< 10),
               wheel(c), large(c,Z), (Z >= 15, Z =<15),
               distance(b,c,W), (W >= 10, W =< 10).
  ```

- **modification of an existing numerical constraint.** This phase proceeds according to the situation at hand.
  1. *positive observation not covered - generalization of a numerical constraint.* Let *Constr* be the numerical constraint that has to be modified, specifically generalized, due to a misclassification of a positive observation. This situation arises because at least one of the numerical values, that is a property of an object in the positive observation, does not fulfill the numerical constraint in the clause. Let v_i be such a numerical value. As we known, the numerical constraint *Constr* is a disjunction of numerical intervals: $Constr = [Range_1; Range_2; \ldots; Range_m]$; the situations that can hold are the following:
     - v_i is included between two intervals

       if $(Range_j = (a_j, b_j)$ and $Range_{j+1} = (a_{j+1}, b_{j+1}))$ and $v_i \in [b_j, a_{j+1}]$ for some $j = 2, \ldots, m-1$ **then**

       $Range_j = (a_j, v_i)$ iff $|v_i - b_j| \leq |v_i - a_{j+1}|$ OR

       $Range_{j+1} = (v_i, b_{j+1})$ iff $|v_i - b_j| \geq |v_i - a_{j+1}|$
     - v_i is a value lower than the first value of the first interval

       if $Range_1 = (a_1, b_1)$ and $v_i \in]-\infty, a_1]$ **then** $Range_1 = (v_i, b_1)$
     - v_i is a value greater than the last value of the last interval

       if $Range_m = (a_m, b_m)$ and $v_i \in [b_m, +\infty[$ **then** $Range_m = (a_m, v_i)$
  2. *negative observation covered - specialization of a numerical constraint.* Let *Constr* be the numerical constraint to be modified, specifically specialized, due to a misclassification of a negative observation. This situation arises because

at least one of the numerical values, that is a property of an object in the negative observation, fulfills the numerical constraint in the clause. Let v_i be such a numerical value. As we known, the numerical constraint $Constr$ is a disjunction of numerical intervals: $Constr = [Range_1; Range_2; \ldots; Range_m]$; the situations that can hold are the following:

- v_i satisfies one of the intervals:

```
    if $Range_j = (a_j, b_j)$, $v_i \in (a_j, b_j)$ for some $j = 1, \ldots, m$ then
      $Range_j = (a_j, v_i[ \vee ]v_i, b_j)$
```

The modification phases, generalization and specialization of numerical intervals, refer reciprocally. In this way, the resulting interval is not affected by the order of example sequence[1]. The addition phase could refer the modification phase that operates on the new numerical constraint just introduced in the clause. The stop criterion in these mutual references is the restoring of the theory correctness with respect to the previous examined observations. The schema reported in the following sketched the integration of the *numerical operators* above defined with the classical inductive operators.

- Inductive Upward Refinement (generalization phase) - Let be C the clause that does not cover the positive example, then:
  a) eliminate one (or more) symbolic literal(s) from C;
  b) eliminate one (or more) numerical literal(s), with the relative numerical constraints, from C;
  c) modify (generalization) the numerical constraints in C;
  d) add a new clause to the theory - the observation itself with symbolic constants properly turned into variables and with numerical constants properly associated to a variable on which a numerical constraint is imposed.

  The steps b) and c) concern the numerical refinement. The procedure described in b) is simultaneously activated with the procedure described in a) allowing the elimination of both symbolic and numerical literals. However, if a numerical literal is eliminated from the description of the clause, then the associated numerical constraints have to be eliminated too. If only symbolic literals are eliminated by the procedure and the correctness of the theory is not restored because of the presence of numerical constraints, then step c) is activated starting from the positive observation under examination and from all the previously negative observations seen. If it is not possible to modify the numerical constraints, they are removed from the description of the clause with their associated numerical literals.
- Inductive Downward Refinement (specialization phase) - Let be C the clause that covers the negative observation, then:
  a') add one (or more) symbolic literal(s), coming from the past positive observations to C;

[1] However, the integration of these operators with the inductive symbolic ones, will suffer from the phenomenon of ordering effects – a phenomenon occurring when different orderings of the same training set give out different results.

b') add one (or more) numerical literal(s), coming from the past positive observations to C;
c') modify (specialization) the numerical constraints in C;
d') add the negation of a symbolic literal, coming from the negative observation that caused the misclassification, to C.

The specialization phase starts with the identification of the clause that erroneously covers the negative observation and analyzes the cause of the misclassification. Indeed, the misclassification of the observation can be due to the symbolic literals or to the numerical ones, i.e. to the violation of the numerical constraints. After this information is gained, the procedure continues with the proper step: if the numerical constraints are violated, step c') is activated with the specialization of the constraints, otherwise new (symbolic/numerical) literals are added to the clause (steps a') and b')). It is important to note that if a numerical literal is added then new numerical constraints, relative to the numerical variables of the literal, have to be created, modified and added to the clause as well.

4 Experiments

The numerical operators and their integration with the symbolic ones embedded in INTHELEX have been tested in the task concerning the extraction of rules for the automatic identification of logical components in scientific paper documents belonging to two different series (the Proceedings of the International Conference on Machine Learning, *ICML*, and the paper formatted according to the Springer Verlag Lectures Notes style, *SVLN*). The results were compared to those of the state-of-art ILP batch learning system Progol [15]. The first-order representation of the layout of such documents is made up of descriptions of the layout blocks that make up a paper document along with their size (height and width) and position (horizontal and vertical) in pixel, type (text, line, picture and mixed) and relative position (horizontal/vertical alignment, adjacency). All the experiments were performed according to a 10-fold cross validation methodology. For each class of documents, the layout blocks that are semantically significant for indexing/retrieval purposes were identified and annotated by expert users, and subsequently used as examples to learn rules for automatically recognizing them when new documents become available. Note that different document classes have different labels, as reported in the following (in square brackets the corresponding number of positive and negative instances is reported). The semantic labels of interest recognized by the domain experts for class *ICML* were: *abstract* [28+,340-], *author* [36+, 332-], *page_number* [27+, 341-] and *title* [29+, 339-]. As regards class *SVLN* the following labels characterizing the objects belonging to it were provided: *abstract* [32+,250-], *affiliation* [30+, 252-], *author* [30+, 252-] and *title* [30+,252-]. Each positive example for a label class to be learned was considered as negative for the others. Furthermore, any document block not labelled by the expert as significant was considered negative for all the components to be learned. Table 1 reports the averaged results as regards number of clauses

Table 1. Understanding in scientific papers domain

	Clauses	Lgg's	Runtime (sec.)	Acc. %
ICML				
abstract	2.42	10.33	753.04	97.84
author	2.81	13.24	4055.14	97.66
page_number	2.48	12.24	1929.99	97.02
title	2.30	10.21	425.35	98.04
SVLN				
abstract	2.9	13	1800.61	94.93
affiliation	3.70	12.33	5285.94	94.52
author	4.48	13.96	7815.58	94.31
title	3.33	12.57	2119.14	94.14

defining the concept (Cl), number of performed generalizations (Lgg), *Runtime* (in seconds) and Predictive Accuracy (Acc.). The overall outcomes reveal that the system was actually able to learn significant definitions for the layout blocks of interest in the documents. Indeed, the predictive accuracy is always very high reaching even 98.04% (never falls below 94.14%).

Figure 1 shows the definitions learned for the layout components of the *ICML* documents in one of the 10 folds. As shown, the learned rules have a high degree of understandability for human experts, thus it is possible to exactly recognize and map on a sample document the layout blocks referred to in the rules, e.g. the domain expert recognized block C, in the rule defining *logic_type_abstract*, as that containing the title (word) "Abstract" in a paper. The performance of the system were compared to that obtained by the Progol batch system (Table 2). For pairwise comparison a 10-fold cross validation paired t-test was used in order to evaluate the difference in effectiveness of the rules induced by the two systems according to the predictive accuracy metric. Requiring a significance level of $\alpha = 0.975$, the test revealed no statistically significant differences among them. Such a comparison turns out encouraging on the goodness of the proposed numerical operators. Indeed, Progol is an ILP system that exploits a batch strategy, and hence it starts the learning process with all the observations available, which facilitates the operations on numerical data.

5 Conclusion and Future Works

In this paper we faced the problem of handling numerical information in an incremental setting exploiting first-order logic as a representation language. The limitations of relational learning systems in dealing with numerical data has been widely shown in the literature, the incremental nature of most real-world application domains further complicates the problem. We proposed a strategy to face these problems, and discussed its integration with classical inductive refinement operators. The strategy was embedded in the ILP system INTHELEX, and experiments carried out on a real-world domain showed good system performance,

```
logic_type_abstract(A) :-
   part_of(B,A),type_of_text(A),
   width(A,I),(I>=190,I=<210),
   part_of(B,C),type_of_text(C),
   width(C,D),(D>=79,D=<489),
   height(C,E),(E>=5,E=<15),
   x_pos_centre(C,F),(F>=7,F=<558),
   part_of(B,G),type_of_text(G),
   width(G,H),(H>=45,H=<54),
   on_top(G,A).
logic_type_author(A) :-
   part_of(B,A),type_of_text(A),
   height(A,E),(E>=42,E=<71),
   width(A,F),(F>=109,F=<234),
   y_pos_centre(A,I),(I>=173,I=<272),
   part_of(B,C),type_of_text(C),
   on_top(D,A),
   alignment_center_col(G,D),
   width(G,H),(H>=502,H=<512).
```

```
logic_type_page_number(A):-
   type_of_text(A),part_of(B,A),
   part_of(B,C),type_of_hor_line(C),
   width(C,D),(D>=16,D=<512),
   height(C,E),(E>=1,E=<5),
   x_pos_centre(C,F),(F>=22,F=<343),
   width(A,G),(G>=9,G=<16),
   height(A,H),(H>=7,H=<8).

logic_type_title(A) :-
   type_of_text(A),
   width(A,F),(F>=152,F=<505),
   part_of(B,A),part_of(B,C),
   type_of_text(C),part_of(B,D),
   on_top(A,D),
   x_pos_centre(D,E),(E>=291,E=<348).
```

Fig. 1. Learned definitions for ICML layout components

even compared to a state-of-art batch system. Future work will concern the optimization of such operators by exploiting statistical measures in the addition and modification phase of numerical constraints and extensive experimentation on different application domains.

References

[1] E. Alphonse and C. Rouveirol. Lazy propositionalisation for relational learning. In W. Horn, editor, *Proceedings of ECAI00*, pages 256–260. 2000.

[2] C. Baroglio, A. Giordana, M. Kaiser, M. Nuttin, and R. Piola. Learning controllers for industrial robots. *Machine Learning*, 23:221–250, 1996.

[3] M. Botta and A. Giordana. Smart+: A multi-strategy learning tool. In *Proceedings of IJCAI93*, pages 937–943, 1993.

[4] M. Botta and R. Piola. Refining numerical constants in first order logic theories. *Machine Learning*, 38:109–131, 2000.

[5] R.M. Cameron-Jones and J.R. Quinlan. Efficient top-down induction of logic programs, 1994.

[6] J. Catlett. On changing continuous attributes into ordered discrete attributes. In Y. Kodratoff, editor, *Proceedings of the Fifth European Working Conference on Learning*, volume 482 of *LNCS*, pages 164–178. Springer Verlag, 1991.

[7] F. Divina, M. Keijzer, and E. Marchiori. A method for handling numerical attributes in GA-based inductive concept learners. In *Genetic and Evolutionary Computation*, volume 2723 of *LNCS*, pages 898–908. Springer-Verlag, 2003.

[8] J. Doungherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In A. Prieditis and S. Russell, editors, *Proceedings of ICML95*, pages 194–202. Morgan Kaufmann, 1995.

Table 2. Comparison with Progol

	Accuracy %		
ICML	INTHELEX	PROGOL	t VALUE
abstract	97.84	99.45	1.97
author	97.66	98.09	0.48
page_number	97.02	97.26	0.23
title	98.04	97.79	-0.27
SVLN			
abstract	94.93	95.02	0.05
affiliation	94.52	91.81	-1.53
author	94.31	91.13	-1.60
title	94.14	96.84	1.54

[9] T. Elomaa. General and efficient multisplitting of numerical attributes. *Machine Learning*, 36:201–244, 1999.

[10] F. Esposito, N. Fanizzi, S. Ferilli, and G. Semeraro. Ideal theory refinement under object identity. In *Proceedings of ICML00*, pages 263–270, 2000.

[11] F. Esposito, S. Ferilli, N. Fanizzi, T.M.A. Basile, and N. Di Mauro. Incremental multistrategy learning for document processing. *Applied Artificial Intelligence*, 17(8/9):859–883, 2003.

[12] F. Esposito, D. Malerba, and V. Marengo. Inductive learning from numerical and symbolic data: An integrated framework. *Intelligent Data Analysis*, 5:445–461, 2001.

[13] U.M. Fayyad and K.B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of IJCAI93*, pages 1022– 1027. Morgan Kaufmann, 1993.

[14] N. Lavrac and S. Dzeroski. *Inductive Logic Programming:Techniques and Applications*. Ellis Horwood, Chichester, 1994.

[15] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[16] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[17] M. Sebag and C. Rouveirol. Constraint inductive logic programming. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 277–294. 1996.

[18] G. Semeraro, F. Esposito, D. Malerba, N. Fanizzi, and S. Ferilli. A logic framework for the incremental inductive synthesis of datalog theories. In N.E. Fuchs, editor, *Proceedings of LOPSTR98*, volume 1463 of *LNCS*, pages 300–321, 1998.

[19] V. Tresp, J. Hollatz, and S. Ahmad. Network structuring and training using rule-based knowledge. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in neural information processing systems 5*, pages 871–878. Morgan Kaufmann, 1993.

[20] L. A. Zadeh. Knowledge representation in fuzzy logic. In R. R. Yager and L. A. Zadeh, editors, *An Introduction to Fuzzy Logic Applications in Intelligent Systems*. Kluwer Academin, 1992.

[21] J.-D. Zucker and J.-G. Ganascia. Learning structurally indeterminate clauses. In D. Page, editor, *Proceedings of ILP98*, volume 1446 of *LNAI*, pages 235–244, 1998.