# Handling Continuous Data in Top-Down Induction of First-Order Rules

Donato Malerba Floriana Esposito Giovanni Semeraro Sergio Caggese
Dipartimento di Informatica, Università degli Studi di Bari
via Orabona 4, 70125 Bari, Italy
{malerba | esposito | semeraro | caggese}@lacam.uniba.it

**Abstract.** Handling numerical information is one of the most important research issues for practical applications of first-order learning systems. This paper is concerned with the problem of inducing first-order classification rules from both numeric and symbolic data. We propose a specialization operator that discretizes continuous data during the learning process. The heuristic function used to choose among different discretizations satisfies a property that can be profitably exploited to improve the efficiency of the specialization operator. The operator has been implemented and tested on the document understanding domain.

## 1 Background and motivation

One of the most important research issues for practical applications of first-order learning systems is the handling of numerical information [7,18]. Indeed, in many applications, continuous data are predominant and should be treated suitably.

Image analysis and understanding is one of the widest application fields where the capability of handling both symbolic and numeric descriptors (that is, attributes and relations) is essential for the successful application of machine learning techniques. In a previous work [9] the authors proposed the application of machine learning techniques to the problem of document understanding, that is, identifying semantically relevant components in the layout extracted from document images. Preliminary results were encouraging, but not totally satisfying. One of the main issues seemed to be the *a priori* discretization of numeric attributes, such as height, width and position of a block. In fact, the first-order learning system used to automatically learn classification rules was not able to perform an on-line, autonomous discretization of numerical descriptors. As pointed out by Connel and Brady [4], both numeric and symbolic descriptions are essential to generate models of visual objects, since the former increase the sensitivity while the latter increase the stability of the internal representation of visual objects.

These considerations prompted the investigation of an extension of our first-order learning algorithm in order to handle numerical descriptors as well. Our goals are:

1. On-line discretization of numerical descriptors should be performed by a specialization operator, since the learning algorithm performs a general-to-specific (or top-down) search.

2. The operator should always guarantee to cover the *seed* example that guides the induction process.

3. The heuristic function used to choose among different discretizations should satisfy a property that reduces the computational complexity of the operator.

In this paper, we present a new specialization operator developed according to these goals. It can specialize a clause by adding literals of the type $f(X_1, \ldots, X_n) \in [a..b]$. An information-theoretic heuristic is used to choose among different discretizations. A property of such a heuristic is claimed and exploited to improve the efficiency of the closed interval selection. This operator has been embedded into a first-order learning system called INDUBI/CSL [20]. The application to the domain of document understanding proved the importance of handling both numeric and symbolic descriptions for this task.

## 2 The representation language

The inductive learning problem solved by INDUBI/CSL is the usual induction of a set of hypotheses $H_1, H_2, \ldots, H_r$ from a set $E$ of training examples. Each hypothesis $H_i$ is the description of a concept $C_i$, $i = 1, \ldots, r$.

The representation language adopted in the system is an evolution of the logic language $VL_{21}$ [21], with two distinct forms of *literals* (also called *selectors*):

$f(t_1, \ldots, t_n)=\text{Value}$ (*simple literal*) and $f(t_1, \ldots, t_n) \in \text{Range}$ (*set literal*)

where f is an *n*-ary function symbol, called *descriptor*, $t_i$'s can be either variable or constant terms, Value is the value taken by f when applied to $t_1, \ldots, t_n$, and Range is a set of possible values taken by f. Some examples of literals are the following: $color(x_1)=red$, $height(x_2) \in [1.1 .. 2.1]$, $distance(x_1, x_2) \in [0.0 .. 1.0]$, and $ontop(x_1, x_2)=true$.

The last example points out the lack of predicate symbols in this logic language. The literals $p(x_1,x_2)$ and $\neg p(x_1,x_2)$ will be represented as $f_p(x_1,x_2)=true$ and $f_p(x_1,x_2)=false$, respectively, where $f_p$ is the function symbol associated to the predicate p. Therefore, INDUBI/CSL can deal with *classical negation*, $\neg$, not with *negation by failure*, not [19].

Literals can be combined to form *definite clauses*, which can be written as:

$$L_0 :- L_1, L_2, \ldots, L_m$$

where the simple literal $L_0$ is called *head* of the clause, while the conjunction of simple or set literals $L_1, L_2, \ldots, L_m$ is named *body*. Definite clauses of interest for classification problems satisfy two different constraints: *linkedness* [15] and *range-restrictedness* [5].

Each training example is represented as a single ground, linked and range-restricted definite clause. On the contrary, a hypothesis *H* is a set of linked, range-restricted definite clauses, called *rule*, such that all clauses have the same head and no constant terms. Permitted literals in *H* can be either single-valued or range-valued. Each concept to be learned has its own *hypothesis language*, that is a set of function symbols used in the body and in the head of clauses defining the concept. Indeed, a peculiarity of INDUBI/CSL is the possibility of learning *multiple*, possibly dependent, concepts [20].

Regardless of the representation language used, a key part of the induction process is the search through a space of hypotheses. A generalization model provides a basis for organizing this search space, since it establishes when a hypothesis *H* entails or *covers* an observation and when an inductive hypothesis is more general/specific than another [3]. The generalization model adopted in INDUBI/CSL is $\theta_{OI}$-subsumption [29], a variant of the well-known $\theta$-subsumption [25]. Under this generalization model, a definite clause C is a generalization of another clause C', if it is obtained from C' by simply applying two distinct operators: *Drop-literal* and *turn-constants-into-variables*. The latter operator turns distinct constants into distinct variables, and replaces all occurrences of a constant $t_i$ with the same variable X (*simple inverse substitution* [28]).

## 3 Learning a single concept

At the high level INDUBI/CSL implements a separate-and-conquer search strategy to generate a rule. The *separate* stage of the algorithm is a loop that checks for the completeness of the current rule and, if this check fails, begins the search for a new consistent clause. The search space for the separate stage is the space of rules. In contrast, the search space of the *conquer* stage is the set of clauses. The conquer stage performs a general-to-specific beam-search to construct a new consistent, linked and range-restricted clause. A thorough description is in [20].

The separate-and-conquer search strategy is adopted in other well-known learning systems, such as FOIL6.2 [27]. On the other hand INDUBI/CSL bases the conquer phase on the concept of *seed* example, whose aim is to guide the learning process. Indeed, if $e^+$ is a positive example to be explained by a hypothesis $H$, then $H$ should contain at least one clause $C$ that generalizes $e^+$. As mentioned above, this means that $C$ is obtained from $e^+$ by applying the *drop-literal* and *turn-constants-into-variables* operators.

In the conquer stage, INDUBI/CSL starts with a seed example $e^+$ and generates a set *Cons* of at least $M$, if any, distinct range-restricted clauses which are consistent and cover $e^+$. The best generalization is selected from *Cons* according to a user-defined preference criterion, and the positive examples covered by such a generalization are removed from the set of examples to be covered. If there are still some positive examples to be covered, a new seed will be selected and a new set *Cons* will be generated. The $M$ clauses are searched in the *specialization hierarchy* rooted into a definite clause with empty body,

$$f(X_1, \ldots, X_n) = \mathrm{Value} :-$$

obtained by applying the operator *turn_constants_into_variables* to the head of $e^+$.

During the specialization process, INDUBI/CSL considers only a subset of $N$ literals in the example: They are chosen according to the associated cost of the main function symbol, so that the user can express a preference for some literals. Obviously, literals that cause the partial clause to become unlinked are not considered at all. All specialized clauses, which cover $e^+$ and possibly other positive examples, are ranked according to a preference criterion. The first $P$ generalizations are selected for the next specialization step. Consistent generalizations are copied into *Cons*.

The problem of finding a definite clause consistent with a sequence of positive and negative examples is a generalization of the problem of finding a consistent conjunctive existential concept [14]. Indeed a definite clause with a nullary predicate in the head:

$$L_0 :- L_1, L_2, \ldots, L_m$$

can be reformulated as:

$$L_0 :- \exists * L_1 \wedge L_2 \wedge \ldots \wedge L_m$$

where $\exists * L_1 \wedge L_2 \wedge \ldots \wedge L_m$ is an existential conjunctive concept. Haussler proved that learning consistent conjunctive existential concepts is an NP-complete problem, under the assumption that the number of parts of each example is upper bounded. Thus, under the same assumption, the general problem of finding a consistent definite clause is NP-complete as well. However, when only one example $e^+$ is positive, while the others are negative, it is possible to find a consistent clause, if any, in a polynomial time. This is done by turning all contants in $e^+$ into variables and then by checking whether the generated clause still covers a negative example.Obviously we are more interested in finding general consistent clauses than in covering only $e^+$, thus it is necessary to perform a search through the space of all clauses that cover the seed example (*seed covering problem*).

In the case of symbolic descriptors, it can be easily shown that the size of this space is exponential in the number of literals of the seed.[1] Indeed, all clauses in the specialization hierarchy can be obtained by adding a (possibly emply) set of literals to

$$f(X_1, \ldots, X_n) = Value :-$$

Since literals used in the specialization process must be generalizations of literals in *body*$(e^+)$, obtained by turning constants into variables, the number of clauses is $2^{|body(e+)|}$.

Nevertheless, INDUBI/CSL explores only a polynomially bounded portion of this space. More precisely, during the first step at most $N$ clauses will be considered. At worst, $N = |\,body(e^+)\,|$. Soon afterwards, $P$ of them are selected for the next specialization step. During the second step each selected hypothesis can be specialized in at most $|\,body(e^+)\,| - 1$ different ways. In general, at the *i*th step, each selected hypothesis can be specialized in at most $|body(e^+)| - i - 1$ different ways. To sum up, the number of generated clauses is:

$$\left|body(e^+)\right| + \sum_{i=1}^{\left|body(e^+)\right|-1} P \cdot i = \left|body(e^+)\right| + \frac{P}{2}\left|body(e^+)\right| \cdot (\left|body(e^+)\right| - 1)$$

This analysis confirms the efficiency of the system while searching for a consistent clause, since the number of tested hypotheses is polynomial in the beam of the search and in the maximum number of literals of a training example. Neither the arity of the function symbols nor the number of variables in any learned clause affect the cost of the beam-search in INDUBI/CSL, which happens with other well-known learning systems, such as FOIL. Indeed, a theoretical analysis of FOIL performed by Pazzani and Kibler [22] showed that the number of tested hypotheses depends on the maximum number of variables in any clause of the learned rule. Such a number grows exponentially with the largest arity of considered predicates.

To sum up, at worst INDUBI/CSL solves as many seed covering problems as the number of positive examples. In this way it bypasses the NP-complete problem of learning a consistent clause, if any, that covers all positive examples.

Finally, it is worth noting that the consistent clause efficiently found by INDUBI/CSL in the conquer stage is not (probabilistically) guaranteed to perform well on further random examples drawn from the instance space according to the same fixed distribution used to build the training set. The algorithm used in the conquer stage is not Probably Approximately Correct (PAC) [31], that is, there is no guarantee that, using only polinomial computation time and polynomial sample size, it will find a consistent clause, if any, with error at most $\varepsilon$ with probability at least $1-\delta$. Haussler [14] has already shown that existential conjunctive concepts are not PAC-learnable unless RP=NP, where RP is the class of problems that have randomized polynomial time algorithms. Thus it is strongly suspected that no PAC algorithm exists for the more general problem of learning a definite clause.

---

1. Here we are not considering inductive biases such as linkedness and range-restrictedness. In general, the introduction of these constraints leads to smaller search spaces.

## 4 Specializing a clause

In order to specialize a clause $G$, INDUBI/CSL has to choose some literals to be added. Both numeric and symbolic data are handled in the same way (see Figure 1). The only difference is that numeric literals already present in $G$ can be reconsidered later on. In this case, the best (sub-)interval is recomputed, since it may be influenced by the addition of further literals. All selected literals are generalizations of literals in the seed, $e^+$, obtained by turning distinct constants into distinct variables.

The computation of the interval for numerical literals is described in Figure 2. A table associated to the term $f(X_1, X_2, \ldots, X_n)$ is built by matching the specialized clause

$$G': \quad G, f(X_1, X_2, \ldots, X_n) \in [-\infty .. +\infty]$$

against positive and negative examples. Each example produces as many entries as the number of matching substitutions. More precisely, if $G''$ is the set of literals in the body of $G'$, that contain some *local* variables not occurring in the head of $G'$, then the number of matching substitutions is upper bounded by two combinatorial functions of the number of local variables and literals in $G''$, respectively. The table, initially empty, contains pairs $\langle$*Value, Class*$\rangle$, where *Class* can be either + or – according to the sign of the example *e* from which *Value* is taken. The *Value* is determined by considering the literal of the example *e* that matches with $f(X_1, X_2, \ldots, X_n) \in [-\infty .. +\infty]$.

Then the problem is finding the interval that best discriminates positive from negative examples. Any threshold value $\alpha$ lying between two consecutive distinct values defines two disjoint intervals: The left interval $[l_1, l_2]$ and the right interval $[r_1, r_2]$. The lower bound $l_1$ of the left interval is the smallest value in the table with a + sign, while the upper bound $l_2$ is the largest value in the table that does not exceed the threshold $\alpha$. On the contrary, the lower bound $r_1$ of the right interval is the smallest value in the table that exceeds $\alpha$, while the upper bound $r_2$ is the largest value with a + sign. When one of the two intervals contains no positive value, then it is set to *undefined*. However, at least one of the two intervals must be defined, since the table contains at least one + value corresponding to the Seed_value. Not all definite intervals are to be considered, since the specialized clause $G, f(X_1, X_2, \ldots, X_n) \in$ Range for a given Range might no longer cover the

```
procedure choose_best_linked_literals(N,e⁺,G,E⁺,E⁻)
List_linked := Ø
foreach literal Lit in e⁺ do
    [f(X_1, X_2, ..., X_n)=Value] := turn_constants_into_variables(Lit, G, e⁺)
    if [f(X_1, X_2, ..., X_n)=Value] is linked to G then
      if the descriptor f is numeric then
        Set_literal := determine_range([f(X_1, X_2, ..., X_n)=Value],e⁺,G,E⁺,E⁻)
        if Set_literal ≠ nil then add Set_literal to List_linked endif
      else
        if [f(X_1, X_2, ..., X_n)=Value] is not in G then add [f(X_1, X_2, ..., X_n)=Value] to List_linked endif
      endif
endforeach
  sort List_linked on cost
return the first N literals in List_linked
```

**Figure 1.** Algorithm for the choice of the best linked literal

```
procedure determine_range([f($X_1$, $X_2$, ..., $X_n$)=Seed_value],$e^+$,G,$E^+$,$E^-$)
  initialize table T[f($X_1$, $X_2$, ..., $X_n$)]
  foreach example e in $E^+$ do
    foreach substitution $\theta$ such that G,f($X_1$, $X_2$, ..., $X_n$)$\in$[$-\infty$ .. $+\infty$] covers e do
      select the literal [f($X_1$, $X_2$, ..., $X_n$)=Value]$\theta$ of e
      add the tuple $\langle$Value ,+$\rangle$ to the table T[f($X_1$, $X_2$, ..., $X_n$)]
    endforeach
  endforeach
  foreach example e in $E^-$ do
    foreach substitution $\theta$ such that G,f($X_1$, $X_2$, ..., $X_n$)$\in$[$-\infty$ .. $+\infty$] covers e do
      select the literal [f($X_1$, $X_2$, ..., $X_n$)=Value]$\theta$ of e
      add the tuple $\langle$Value ,-$\rangle$ to the table T[f($X_1$, $X_2$, ..., $X_n$)]
    endforeach
  endforeach
  sort table T[f($X_1$, $X_2$, ..., $X_n$)] on the Value field
  Cut := determine_all_cut-points(T[f($X_1$, $X_2$, ..., $X_n$)])
  MaxIG := $-\infty$
  foreach cut-point $\alpha$ in Cut do
    determine the left and right intervals [$l_1$,$l_2$], [$r_1$,$r_2$] with $l_2<\alpha<r_1$
    if Seed_value$\in$[$l_1$ .. $l_2$] then Admissible_interval := [$l_1$, $l_2$] else Admissible_interval := [$r_1$, $r_2$] endif
    IG := information_gain(T[f($X_1$, $X_2$, ..., $X_n$)], Admissible_interval)
    if IG > MaxIG then
      Best_interval:=Admissible_interval
      MaxIG := IG
    endif
  endforeach
  if Max IG $\neq -\infty$ then return [f($X_1$, $X_2$, ..., $X_n$)$\in$Best_interval] else return nil endif
```

**Figure 2.** Choice of the best range for numerical descriptors.

seed example $e^+$. Those definite intervals that include the Seed_value are said to be *admissible*, because they guarantee that the corresponding specializations still cover $e^+$.

The best admissible interval is selected according to an information-theoretic heuristic, the *information gain*, which is an entropic measure commonly used in decision tree induction [26] and in relational learning [27]. By looking at the table as a source of messages labelled + and –, the expected information on the class membership conveyed from a randomly selected message is:

$$info(n^+,n^-) = -\frac{n^+}{n^+ + n^-}log_2\frac{n^+}{n^+ + n^-} - \frac{n^-}{n^+ + n^-}log_2\frac{n^-}{n^+ + n^-}$$

where $n^+$ and $n^-$ are the number of values in the table with a positive and a negative sign, respectively. If we partition the table into two subsets, $S_1$ and $S_2$, the former containing $n_1^+ + n_1^-$ values falling within an admissible interval and the latter containing the remaining values, the information provided by $S_1$ will be close to zero when almost all cases have the same + or – sign. Although the information prefers partitions that cover a large number of cases of a single class and a few cases of other classes, we must bias such a preference towards intervals with a high number of positive cases, as well. The following *weighted entropy*:

$$E(n_I^+, n_I^-) = \frac{n_I^-}{n_I^+} info(n_I^+, n_I^-)$$

penalizes those admissible intervals with a low percentage of positive cases. The quantity

$$gain(n^+, n^-, n_I^+, n_I^-) = info(n^+, n^-) - E(n_I^+, n_I^-)$$

measures the information gained by replacing the table with $S_I$. A heuristic would be choosing the admissible interval that maximizes the information gain, i.e., that minimizes $E(n_I^+, n_I^-)$. This heuristic differs from that adopted in other well-known learning systems. In particular, ID3 [26] does not weight the entropy, while FOIL uses only the information content of the positive class.

As a concrete illustration of the procedure determine_range, consider the table below.

| *Value* | 0.5 | 0.7 | 0.9 | 1.0 | 1.5 | 1.5 | 1.5 | 1.7 | 2.5 | 2.5 |
|---|---|---|---|---|---|---|---|---|---|---|
| *Sign* | + | - | - | - | - | - | + | + | - | + |

There are four possible cut points that generate the following intervals:

| α | *0.60* | *1.25* | *1.60* | *2.10* |
|---|---|---|---|---|
| [l1, l2] | [0.50 .. 0.50] | [0.50 .. 1.00] | [0.50 .. 1.50] | [0.50 .. 1.70] |
| [r1, r2] | [0.70 .. 2.50] | [1.50 .. 2.50] | [1.70 .. 2.50] | [2.50 .. 2.50] |

Let us suppose that Seed_value equals 1.50. Then only those intervals including 1.50 are admissible. The weighted entropy for each of them is

| **Adm. interval** | [0.70 .. 2.50] | [1.50 .. 2.50] | [0.50 .. 1.50] | [0.50 .. 1.70] |
|---|---|---|---|---|
| E | 1.836592 | 1.000000 | 2.157801 | 1.590723 |

Thus, the best interval is the second one, with weighted entropy equal to 1.0.

Note that cut points 0.80 and 0.95 have not been considered. Indeed, only those between two consecutive distinct values with a different sign (*boundary points*) are considered. This choice is due to the following

**Theorem.** If a cut-point α minimizes the measure $E(n_I^+, n_I^-)$, then α is a boundary point.

The proof can be obtained electronically from http://lacam.uniba.it:8000/pagine/proof.html.

This result helps to discard several computations of the gain by considering only boundary points, so improving the efficiency of the procedure *determine_range*. Actually, the theorem above is similar to that proved by Fayyad and Irani [12] for a different measure, namely the "unweighted" class information entropy computed in some decision tree learning systems.

## 5 Other approaches

The first attempt to deal with continuous-valued attributes in first-order systems that learn classification was made by Bergadano and Bisio [1], who proposed a method to automatically set some parameters of predicate "schemes" with a parametric semantics.

Later on, a two-step approach was implemented in the system ML-SMART [2]: First, a tentative numerical parameter is learned, and then a standard genetic algorithm is applied to refine the numerical knowledge. On the contrary, the system Rigel [13] discretizes continuous data by applying the generalization operator called *consistent extending reference rule*, which extends the reference of a selector, that is the set of values taken by a function f. Values are added only if this increases the number of covered positive examples without covering negative ones. A different approach was proposed by Esposito *et al.* [8] who combined a discriminant analysis technique for linear classification with a first-order learning method, so that the numerical information is handled by linear classifiers, while the symbolic attributes and relations are used by the first-order learning system. A common characteristic of all these approaches is that they have been conceived for systems that can learn only rules with nullary predicates in the head, that is with predicates corresponding to propositional classes.

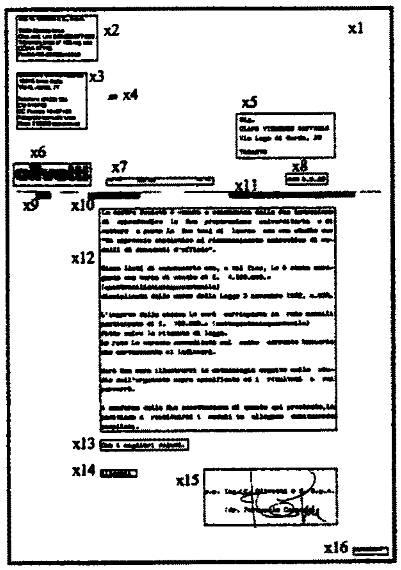
Dzeroski *et al.* [6] proposed transforming first-order representations into propositional form, in order to handle real numbers by means of techniques already tested in zeroth-order induction systems. Nevertheless, the transformation algorithm is applicable only when the background relations are determinate [17].

A different approach has been adopted in FOIL6.2 [27]. The system automatically produces comparative literals of type $V_i > k$, $V_i \leq k$, $V_i > V_j$, $V_i \leq V_j$, where $V_i$ and $V_j$ are numerical variables already present in other non-comparative literals and $k$ is a numerical threshold. The selection of the threshold is based on an information-theoretic measure, which is different from that adopted in our system. Indeed, FOIL's specialization operator does not guarantee it will cover a specific positive example, the seed. Other differences between the two systems concern the top-down learning process (beam-search, seed-driven vs. hill-climbing, information-gain-driven), and the stopping criterion (at least $M$ consistent hypotheses found vs. minimum description length).

Much related work can also be found in other contexts, such as qualitative and relational regression in inductive logic programming, and learning numerical constraints in inductive constraint logic programming. An updated review can be found in [18].

## 6 Application to document understanding

INDUBI/CSL has been applied to some zeroth-order learning problems involving both symbolic and numeric attributes [11]. In this paper we present the application to the field of document understanding. According to the ODA/ODIF standard [16], any document is characterized by two different structures representing both its internal organization and its content: The *layout* (or *geometrical*) structure and the *logical* structure. The former associates the content of a document with a hierarchy of *layout objects*, such as text lines, vertical/horizontal lines, graphic/photographic elements, pages, and so on. The latter associates the content of a document with a hierarchy of *logical objects*, such as sender/receiver of a business letter, title/authors of an article, and so on. The term *document analysis* denotes the extraction of the layout structure from the bitmap of a document, while the term *document understanding* denotes the process of mapping the layout structure of a document into the corresponding logical structure [30]. The document understanding process is based on the assumption that documents can be understood by means of their layout structures alone.



```
logic_type(x8)=date :-
   part_of(x1,x2)=true,...,part_of(x1,x16)=true,
   width(x2)= medium, ... , width(x16)= medium_small,
   height(x2)= medium_small, ..., height(x16)= smallest,
   type(x2)= text, ..., type(x16)= text,
   position(x2)= top_left, ..., position(x16)= bottom_right,
   ontop(x2,x3)=true, ..., ontop(x13,x14)=true,
   toright(x6,x7)=true, ... ,toright(x9,x10)=true,
   aligned(x2,x3)=only_left_col, ...,
   aligned(x13,x14)=only_left_col

logic_type(x8)=date :-
   part_of(x1,x2)=true,..., part_of(x1,x16)=true,
   width(x2)= 120.0, ..., width(x16)= 44.0,
   height(x2)= 63.0, ..., height(x16)= 5.0,
   type(x2)= text, ..., type(x16)= text,
   x_pos_center(x2)= 89.0, x_pos_center(x16)=568.0,
   y_pos_center(x2)= 40.0, y_pos_center(x16)= 836.0,
   ontop(x2,x3)=true, ... , ontop(x13,x14)=true,
   toright(x6,x7)=true, ..., toright(x9,x10)=true,
   aligned(x2,x3)=only_left_col, ...,
   aligned(x13,x14)=only_left_col
```

**Figure 3.** Layout of a business letter and the symbolic (up), numeric/symbolic (down) descriptions of the logical component *date*.

The mapping of the layout structure into the logical structure can be represented as a set of rules. Traditionally, such rules have been handcoded for particular kinds of documents [22], requiring much human tune and effort. We proposed the application of inductive learning techniques in order to automatically generate the rules from a set of training examples [9]. The user-trainer is asked to label some layout components of a set of training documents according to their logical meaning. Those layout components with no clear logical meaning are not labelled. Therefore, each document generates as many training examples as the number of layout components. Classes of training examples correspond to the distinct logical components to be recognized in a document. The unlabelled layout objects play the role of counterexamples for all the classes to be learned.

Each training example is represented as a definite ground clause, where different constants represent distinct layout components of a page layout. The choice of a representation language for the description of the layout of each document is very important. In previous experiments we used only symbolic descriptors by discretizing numeric attributes, such as height, width and position of a block (see Figure 3). Since the current release of INDUBI/CSL is able to handle numerical descriptors as well, we decided to organize an experiment to test the improvement of the generated rules in terms of accuracy, learning time and simplicity.

We considered a set of 30 business letters containing 364 layout objects. Each document was described with only symbolic descriptors or mixed numeric/symbolic descriptors. Experimental results for a 10-fold cross-validation are summarized below:

| **Average Number of Errors** | | **p-value Wilcoxon signed ranks test** | **Number of Clauses** | | **Average Learning Time** | |
|---|---|---|---|---|---|---|
| **symbolic** | **mixed** | | **symbolic** | **mixed** | **symbolic** | **mixed** |
| 3.6 | 2.9 | 0.3114 | 28.0 | 11.5 | 20:24 | 19:17 |

*logic_type*($X_1$)=*date* ← *y-pos-centre*($X_1$) ∈[262.0..279.0], *x-pos-centre*($X_1$) ∈[453.0 ..525.0]

*logic_type*($X_1$)=*date* ← *width*($X_1$)= *medium_small*, *height*($X_1$)= *very_very_small*,
    *aligned*($X_2$,$X_1$)=*both_rows*
*logic_type*($X_1$)=*date* ← *position*($X_1$)= *top_right*, *ontop*($X_1$,$X_3$)=*true*, *toright*($X_2$,$X_3$)=*true*,
    *aligned*($X_2$,$X_3$)=*both_rows*, *aligned*($X_4$,$X_2$)=*only_left_col*
*logic_type*($X_1$)=*date* ← *width*($X_1$)= *medium*, *ontop*($X_1$,$X_3$)=*true*, *aligned*($X_2$,$X_1$)=*both_rows*
*logic_type*($X_1$)=*date* ← *aligned*($X_4$,$X_2$)=*both_rows*, *aligned*($X_3$,$X_4$)=*only_lower_row*,
    *aligned*($X_2$,$X_1$)=*both_rows*
*logic_type*($X_1$)=*date* ← *aligned*($X_3$,$X_2$)=*both_rows*, *aligned*($X_2$,$X_1$)=*only_lower_row*
*logic_type*($X_1$)=*date* ← *toright*($X_4$,$X_2$)=*true*, *aligned*($X_3$,$X_4$)=*only_left_col*,
    *aligned*($X_2$,$X_1$)=*only_lower_row*
*logic_type*($X_1$)=*date* ← *width*($X_1$)=*small*, *ontop*($X_1$,$X_2$)=*true*,*aligned*($X_3$,$X_1$)=*only_right_col*

**Figure 4.** Rule for the logic type *date* learned in one of the ten trials from mixed numeric-symbolic (up) and only symbolic descriptors (down).

The average number of errors made by the new release of INDUBI/CSL is decreased, although not significantly w.r.t. the non-parametric Wilcoxon signed-ranks test [23]. However, by decomposing the average number of errors into omission and commission errors we can conclude that rules generated from mixed descriptions made a significantly lower number of commission errors[2] (0.3 vs. 1.5, p-value=0.0277), and slightly increased the number of omission errors (2.6 vs. 2.1, p-value=0.7353). On-line discretization of numerical attributes increases the sensitivity of the learner, thus reducing commission errors, but increasing omission errors. Indeed, rules containing literals of the type $f(X_1, \ldots, X_n) \in [a..b]$ often miss the match with an instance because the value taken by f is either a little higher than b or a little lower than a. Although commission and omission errors are generally considered equally important, it is worthwhile to observe that in document processing systems omission errors are deemed to be less serious than commission errors, which can lead to totally erroneous storing of information. Moreover, we have also shown that a significant recovery of omission errors can be obtained by relaxing the definition of flexible matching between definite clauses [10].

As to the other parameters, we observed that the introduction of numerical descriptors simplified the classification rules (see an example of learned rules in Figure 4) and reduced the learning time. The latter result is somehow surprising since the symbolic description of a training instance is slightly simpler than the corresponding symbolic/numeric description. This strengthened our opinion that the handling of numerical attributes was actually beneficial in this application.

As a final experiment, we compared INDUBI/CSL to FOIL6.2 on the mixed approach, since Quinlan's system has been often used as a yardstick for other empirical studies on first-order inductive learning. The results are summarized below.

| **Average Number of Omission/Commission Errors** | | **Average Number of Errors** | | **Number of Clauses** | |
|---|---|---|---|---|---|
| **INDUBI/CSL** | **Foil6.2** | **INDUBI** | **Foil6.2** | **INDUBI** | **Foil6.2** |
| 2.6/0.3 | 1.3/1.5 | 2.9 | 2.8 | 11.5 | 11.0 |

2. Omission errors are made when some examples of $C_i$ are not classified as instances of $C_i$. Commission errors are made when some examples of $C_i$ are classified into $C_j$, with $j \neq i$.

The average error rate, as well as the average number of clauses, is almost the same for both systems. The difference is in the type of errors made by the two systems. Foil6.2 discretizes into larger intervals than INDUBI/CSL, thus causing more commission errors than our system.

## 7 Conclusions

Handling both numeric and symbolic descriptors is an important issue for the successful application of first-order learning systems to real-world problems. In this paper we have presented a specialization operator for the on-line discretization of continuous data. Such an operator has been implemented in the learning system INDUBI/CSL and tested in the field of document understanding. In particular, we observed that the on-line discretization of numerical attributes increases the sensitivity of the learner, thus reducing commission errors and simultaneously increasing omission errors. A solution to this problem can come from a definition of flexible matching between definite clauses. Furthermore, the definition of a distance measure between examples can help to solve the problem of choosing the seed that guides the induction process.

## Acknowledgments

The authors would like to thank Marzio Cristiano Rotondo for his valuable collaboration in conducting the experiments, Lynn Rudd for her help in re-reading the first draft of the paper, and the anonymous reviewers for their useful comments.

## References

[1] F. Bergadano, and R. Bisio. Constructive learning with continuous-valued attributes. In B. Bouchon, L. Saitta, and R.R. Yager (Eds.), *Uncertainty and Intelligent Systems.*, LNCS 313, Berlin: Springer-Verlag, pp. 54-162, 1988.

[2] M. Botta, and A.Giordana. Learning quantitative features in a symbolic environment. In Z.W. Ras and M. Zemankova (Eds.), *Methodologies for Intelligent Systems*, LNAI 542, Berlin: Springer-Verlag, pp. 296-305, 1991.

[3] W. Buntine. Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, vol. 36, no. 2, pp. 375-399, 1988.

[4] J.H. Connell and M. Brady. Generating and generalizing models of visual objects. *Artificial Intelligence*, vol. 31, no. 2, pp. 159-183, 1987.

[5] L. De Raedt. *Interactive Theory Revision*. London: Academic Press, 1992.

[6] S. Dzeroski, L. Todorovski, and T. Urbancic. Handling real numbers in ILP: A step towards better behavioural clones (Extended abstract). In N. Lavrac and S. Wrobel (Eds.), *Machine Learning: ECML95*, LNAI 912, Berlin: Springer, pp. 283-286, 1995.

[7] S. Dzeroski, and I. Bratko. Applications of inductive logic programming. In L. De Raedt (Ed.), *Advances in Inductive Logic Programming*, Amsterdam: IOS Press, pp. 65-81, 1996.

[8] F. Esposito, D. Malerba, and G. Semeraro. Incorporating statistical techniques into empirical symbolic learning systems. In D.J. Hand (Ed.), *Artificial Intelligence Frontiers in Statistics*, London: Chapman & Hall, pp. 168-181, 1993.

[9] F. Esposito, D. Malerba, and G. Semeraro. Multistrategy learning for document recognition. *Applied Artificial Intelligence*, vol. 8, no. 1, pp. 33-84, 1994.

[10] F. Esposito, S. Caggese, D. Malerba, and G. Semeraro. Classification in noisy domains by flexible matching. *Proceedings of the European Symposium on Intelligent Techniques*, pp. 45-49, 1997.

[11] F. Esposito, S. Caggese, D. Malerba, and G. Semeraro. Discretizing continuous data while learning first-order rules. In M. van Someren & G. Widmer, *9th European Conference on Machine Learning - Poster Papers*, pp. 47-56, Edicní oddelení VŠE, Prague, 1997.

[12] U.M. Fayyad and K.B. Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, vol. 8, pp. 87-102, 1992.

[13] R. Gemello, F. Mana, and L. Saitta. Rigel: An inductive learning system. *Machine Learning*, vol. 6, no. 1, pp. 7-35, 1991.

[14] D. Haussler. Learning conjunctive concepts in structural domains. Machine Learning, col. 4, no. 1, pp. 7-40, 1989.

[15] N. Helft. Inductive generalization: A logical framework. In I. Bratko and N. Lavrac (Eds.), *Progress in Machine Learning - Proceedings of the EWSL87*, Sigma Press, pp. 149-157, 1987.

[16] W. Horak. Office document architecture and office document interchange formats: current status of international standardization. *IEEE Computer*, vol. 18, no. 10, pp. 50-60, 1985.

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

[18] N. Lavrac, S. Dzeroski, and I. Bratko, Handling imperfect data in inductive logic programming. In L. De Raedt (Ed.), *Advances in Inductive Logic Programming*, Amsterdam: IOS Press, pp. 48-64, 1996.

[19] J.W. Lloyd. *Foundations of Logic Programming*. Second Edition. Berlin: Springer-Verlag, 1987.

[20] D. Malerba, G. Semeraro, and F. Esposito. A multistrategy approach to learning multiple dependent concepts. In C. Taylor and R. Nakhaeizadeh (Eds.), *Machine Learning and Statistics: The Interface*, London: Wiley, pp. 87-106, 1997.

[21] R.S. Michalski. Pattern Recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, no.4, pp. 349-361, 1980.

[22] G. Nagy, S.C. Seth, and S.D. Stoddard. A prototype document image analysis system for technical journals. *IEEE Computer*, vol. 25, no. 7, pp. 10-22, 1992.

[23] M. Orkin, and R. Drogin. *Vital Statistics*, New York, NY: McGraw Hill, 1990.

[24] M.J. Pazzani, & D. Kibler. The utility of knowledge in inductive learning. *Machine Learning*, vol. 9, no. 1, pp. 57-94, 1992.

[25] G.D. Plotkin. Automatic methods of inductive inference. PhD thesis, Edinburgh University, August 1971.

[26] R. Quinlan. Induction of decision trees. *Machine Learning*, vol. 1, pp. 81-106, 1986.

[27] J.R. Quinlan, and R.M. Cameron-Jones. FOIL: A midterm report. In P.B. Brazdil (Ed.), *Machine Learning: ECML-93*, Lecture Notes in Artificial Intelligence, 667, Berlin: Springer-Verlag, pp. 3-20, 1993.

[28] C. Rouveirol. Flattening and saturation: Two representation changes for generalization. *Machine Learning*, vol. 14, pp. 219-232, 1994.

[29] G. Semeraro, F. Esposito, and D. Malerba. Ideal refinement of Datalog programs. In M. Proietti (Ed.), *Logic Program Synthesis and Transformation*, LNCS 1048, Berlin:Springer-Verlag, pp. 120-136, 1996.

[30] Y.Y. Tang, C.D. Yan, and C.Y. Suen. Document processing for automatic knowledge acquisition. *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, no. 1, pp. 3-21, 1994.

[31] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, vol. 27, no. 11, pp. 1134-1142, 1984.