# Background knowledge-enrichment for bottom clauses improving.

Orlando Muñoz Texzocotetla and René MacKinney-Romero

Departamento de Ingeniería Eléctrica
Universidad Autónoma Metropolitana
México D.F. 09340, México

**Abstract.** In this paper we present a method to enrich the hypothesis language which is used to construct the bottom clause. This approach, which is embedded in the Aleph system, is based on numerical fixed subintervals and categorical subsets. Each subinterval/subset contains values, for each attribute which is predefined, that are related with the example to saturate. The enriched language allows to reduce the number of rules of the final theories and, in some cases, to improve the accuracy.

**Keywords:** Artificial Intelligence, ILP, discretization, grouping, numerical attributes, categorical attributes

## 1 Introduction

Most of the ILP algorithms induce one clause at a time in order to find a theory $T$. Each clause in $T$ is usually constructed with a single numerical (or categorical) value for each attribute[1]. This may result in inaccurate theories with a lot of rules. To overcome these drawbacks, some ILP systems use approaches which are based in the following strategies:

**Discretization.** Some ILP systems discretize numerical attributes in a global way, and the final intervals are used during the learning process. TILDE [3] and ECL-GSD [5] follow this strategy. In [9], a global binary discretization method is presented, but in addition to this each categorical attribute is grouped in two subsets. **Propositionalization.** These strategies consist in transforming a relational problem into a propositional problem. The main objective it is to solve a problem that was originally a relational problem, with faster and more efficient propositional algorithms than the relational systems. LINUS and DINUS [7] systems follow that procedure. **Genetic algorithms.** To deal with numerical attributes, some systems implement genetic refinement operators which search globally the best intervals. In [5], several refinement operators are presented, and a genetic algorithm is used to test the intervals. **Numerical Reasoning.** In [11], new relations (equalities, inequalities, regression models, etc.) are added a priori into the background knowledge. These relations are lazily evaluated during the bottom clause construction. Numerical reasoning is improved in [1], since

[1] We refer to the arguments of predicates as attributes.

it proposes to improve noise handling by mean of statistical-based techniques. Regarding the categorical data, we think that using grouping algorithms for categorical attributes can also improve the final theories.

In this paper we present a method to enrich the hypothesis language which is used to construct the bottom clause. This approach deals with both categorical and numerical attributes, in a local or global way. The enriched language allows to reduce the number of rules of the final theories and, in some cases, to improve the accuracy. This paper is organized as follows: section 2 describes the proposed method; section 3 shows the experimental results on several data sets; finally, section 4 presents our conclusions and future work.

## 2 The method

To enrich the hypothesis language our method tests different numerical subintervals (or categorical subsets) according to an evaluation function, then the best qualified are added into the background knowledge. Due to the great amount of subintervals/subsets that can be created we decided to use an evolutive approach to generate and evaluate them. This method was embedded in the basic algorithm of Aleph as follows:

*0. State parameters.* In addition to the mode declarations, types and determinations, users can also declare the attributes to be discretized/grouped in any of the two following ways:

*i.* $lit(\ldots, Attr, \ldots)$, $goal(\ldots, Class)$
Where *Attr* is the attribute to discretize/group respect to $Class$ which can have more than two values. The predicate *lit* is into the background knowledge (it is possible to declare two or more attributes in *lit*), and *goal* is the target predicate.

*ii.* $lit(\ldots, Attr, \ldots)$
In this case the corresponding classes are *positive* and *negative*.

*1. Select an example e.* In this step, a subset of values of *Attr* are related to the example selected. For instance, let $e = goal(a)$ be the example selected and $lit(a, 12)$, $lit(a, 15)$, $lit(a, 20)$, $lit(b, 30)$, $lit(b, 35)$ be five facts in the background knowledge then the values $12, 15, 20$ are related to $e$. If *Attr* is a numerical attribute, then these values form a *FixedSubinterval*. If *Attr* is a categorical attribute, then we call *FixedSubset* to these subset of values. These are fixed because it will invariably be in all subintervals/subsets that will be tested.

*1.5 Discretization/Grouping.* Before saturation each attribute declared by the user is processed. Depending on the attribute type there are two possible cases.

*i. Let Attr be a numerical attribute* and $FixedSubinterval = [min, max]$ be the interval that contains all values in *Attr* that are related to $e$. Then the proposed genetic algorithm returns a set of numerical intervals $I_n$ such that: $|I_n|$ is defined by the user and $I_n = \{x \mid x = [min', max'] \wedge min' \leq min \wedge max \leq max'\}$.

Furthermore each element in $I_n$ represents a chromosome. To evolve the chromosomes several mutation operators are defined. These can enlarge or shrink an interval, either on the left side, on the right side, or on both sides. The

fitness function for a chromosome $x$ is given by the information gain, see Eq. (1). In this case the GA maximizes the information gain to find better subintervals.

$$IG(x) = ent(Int) - \frac{count(x)}{count(Int)} ent(x) \tag{1}$$

The corresponding entropy for some interval $Int$ is:

$$ent(\text{Int}) = -\sum_{j=1}^{|S|} p(Int, j) log_2(p(Int, j))$$

where: $|S|$ = number of classes, $Int$ is the numerical interval of $Attr$, $count(Int)$ = numeric values into interval $Int$, $p(Int, j)$, is the proportion of values in $Int$ that belong to the $jth$ class.

*ii. Let* Attr *be a categorical attribute* and *FixedSubset* be the set of all values related to $e$. The proposed GA searches the best $|C|$ subsets of categorical values with the following constraint: $C = \{y \mid FixedSubset \subseteq y\}$.

A chromosome is represented by a subset $y$. The mutation operators defined add a new categorical value in $y$, delete an element from the chromosome, or swap an element between the chromosome and the rest of categorical values, but maintaining the FixedSubset into each new chromosome. The crossover operator swaps an element between two chromosomes.

In this case, the fitness function is based on the distance between two values of a categorical attribute. The fitness for a chromosome $y$ is the total sum of the distances for each value pair in $y$. In addition to this, if the sum of distances for two or more chromosomes is the same, then the value $\frac{1}{|y|}$ is added, because we want to minimize and to favor the subsets with more values. This fitness function, called DILCA [6], is showed in Eq. (2).

$$Fitness(y) = \sum_{c_i c_j \in y} \sqrt{\sum_{s_t \in S} (P(c_i \mid s_t) - P(c_j \mid s_t))^2} + \frac{1}{|y|} \tag{2}$$

where: $c_i, c_j \in y$ such as $i \neq j$. $S$ is the set of classes defined, $P(c_i \mid s_t)$ is the conditional probability for $c_i$ given $s_t$, and $P(c_j \mid s_t)$ is the conditional probability for $c_j$ given $s_t$.

In both cases the user can give the number of chromosomes to be added.

*2. Build a bottom clause* that entails the example e. The enriched language is used to build a bottom clause. Consider the goal relation $no\_payment(S)$ which is true for those students who do not need to repay its loan. If literals in background knowledge are $unemployed(S)$, $enrolled(S, School, Units)$, where $Units$ is numerical and $School$ is categorical, then a bottom clause would be like this:

```
no_payment(A) :- unemployed(A),enrolled(A,B,C),
interval(C,[11.0,13.4]),interval(C,[11.7,12.5]),interval(C,[11.3,13.3]),
interval(C,[11.3,14.0]),interval(C,[11.0,13.5]),interval(C,[10.0,12.3]),
member(B,[occ,smc,uci,ucla,ucsd]),member(B,[smc,ucb,ucla,ucsd]).
```

*3. Search.* Aleph uses several search strategies (best first, depth first search, etc.), refinement operators, and evaluation functions (as mentioned earlier).

*4. Remove covered (redundant) examples* and add the best clause found to the current theory. Go to step 1.

In the next section we present the experiments performed to compare the accuracy of our method.

## 3 Experiments

In this section we compare our method (we call it *Extended Aleph*) with the standard Aleph, and the lazy evaluation in Aleph [11]. Our method works as the latter, namely with the bottom clause construction as our method. A 10-fold, cross-validation was performed to compare the accuracy and the simplicity (number of rules) of the final theories.

From the UCI machine learning repository [2], we used Australian Credit Approval, Pittsburgh Bridges, Japanese Credit Screening, German Credit, Iris, Moral Reasoner, Ecoli, and Student Loan datasets. We also performed tests over the well-known ILP datasets: Carcinogenesis [12], KRK illegal chess position (KRKi) [8], and mutagenesis [10] (on "regression friendly" data).

All these methods were executed in Yap Prolog version 6.0, with the global parameters: $minpos = 2$, and $noise = 5$; the remaining of the parameters are fixed by default in Aleph. All experiments were performed on a modern multicore PC machine.

**Extended Aleph vs Aleph.** Table 1, shows that our implementation improves the accuracy in most databases, with the exception of two datasets: *Pittsburgh Bridges and Moral Reasoner*. We can also see (figure 1) that the increase of the accuracy for the datasets *Japanese Credit Screening and Iris* is significant, 15% and 7.8% respectively. Regarding the simplicity of the final theories, extended Aleph system does not improve the simplicity with two datasets: *German Credit and KRKi*, but with the other datasets (excepting *Moral Reasoner*) our approach decreases the number of rules in the final theories. In particular, with *Iris* and *Student Loan* the reduction is almost 50% (figure 2).

**Extended aleph vs Lazy Evaluation in Aleph.** As in the previous case, Extended Aleph improves the accuracy except in two datasets: *Moral Reasoner and Student Loan* (table 1). Furthermore, Lazy Evaluation overcomes our method only in one single dataset: *Student Loan*, (figure 2). Although both have the same accuracy.

| **Dataset** | **Accuracy** (%) | | | **Number of rules** | | |
|---|---|---|---|---|---|---|
| | Aleph | Ext. Aleph | Lazy | Aleph | Ext. Aleph | Lazy |
| Australian Credit Approval (Aus) | 82 | **83** | 80.5 | 27.40 | **24.70** | 29.9 |
| Pittsburgh Bridges (Pitts) | 89 | 89 | 88.7 | 13.2 | **13.1** | 13.4 |
| Japanese Credit Screening (Japan) | 53 | **69** | 65 | 13.2 | **8.6** | 11.8 |
| German Credit (German) | 70.9 | **71.2** | 70.8 | **51.4** | 52.8 | 53.4 |
| Iris | 85.9 | **93.7** | 89.1 | 26.9 | **13.7** | 22.6 |
| Moral Reasoner (Moral) | 96 | 96 | 96 | 1 | 1 | 1 |
| Ecoli | 91 | **92** | 91 | 37.3 | **35.4** | 43.5 |
| Student Loan (Student) | 93.7 | **97.6** | **97.6** | 46.6 | 25.7 | **21.5** |
| Carcinogenesis (Carcino) | 58.6 | **61.3** | 56.3 | 21.9 | **18.6** | 28.9 |
| KRKi | 49.4 | **50.4** | 49.4 | **66.7** | 68.1 | **66.7** |
| Mutagenesis (Muta) | 77.1 | **80.8** | 78.3 | 11.2 | **9** | 10.4 |

**Table 1.** Predictive accuracies and number of rules of the analyzed datasets.

In general, Extended Aleph improves or maintains the accuracy (figure 1), and decreases the number of rules for most of datasets. With regard to the running time, Extended Aleph significantly exceeds this measure with Carcinogenesis: Aleph 36.1$s$, Lazy 222.33$s$, Ext. Aleph **698.9s**; and Ecoli: Aleph 11.5$s$; Lazy 63.2$s$; Ext. Aleph **707.5s**.

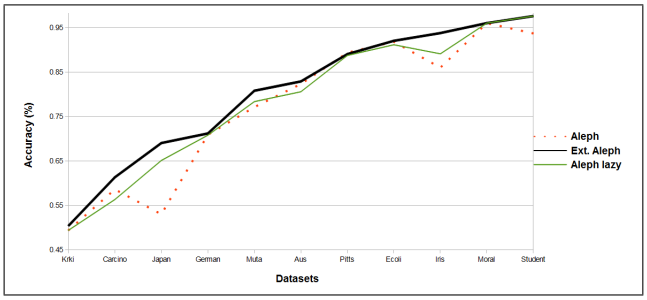


**Fig. 1.** Extended Aleph maintains or improves the accuracy.

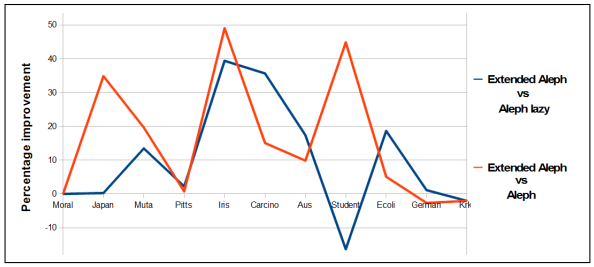


**Fig. 2.** Percentage reduction of the number of rules in final theories.

## 4 Conclusions and Future Work

With the obtained results, we can draw some conclusions: this method does not guarantee to improve the accuracy and simplicity of the theories in all cases. The improvement depends on the selected attributes, namely if an attribute is relevant in the theory construction then its processing will help to improve the final theory. ILP problems whose final theories do not need constants to be induce on them can not benefit from this method. In this case, variables are useful to better explain that concept. The size of the search space can be affected with the proposed method. On one hand a discretized/grouped attribute which is not relevant can generate a large amount of unnecessary candidate rules. On

the other hand a discretized/grouped attribute which is relevant can decrease significantly the number of candidate rules. Users do not know in advance which attributes will be most useful, therefore the intuition is an important element to select each attribute. Our method allows to experiment easily with different attributes. Finally, unlike other genetic approaches like SMART+ [4], our method can deal with both categorical and numerical attributes.

Some considerations for further work are as follows. Since not all cases were successful it is necessary to test other fitness functions and more datasets. These tests will help us to identify the factors which affect both accuracy and simplicity of the final theories, as well as the search space size. Another path of research is to look into multivariable predicates, if there are correlations between two or more attributes. Thus, we want to investigate if these correlations are relevant to improve the performance in ILP (accuracy and simplicity) and what kind of problems can be treated with these predicates. Finally, we want to implement in Aleph system these ideas and compare performance with other ILP systems that handle data types.

## References


1. A. Alves, R. Camacho, and E. Oliveira, *Improving numerical reasoning capabilities of inductive logic programming systems*, in IBERAMIA, 2004, pp. 195–204.
2. K. Bache and M. Lichman, *UCI machine learning repository*, 2013.
3. H. Blockeel and L. D. Raedt, *Lookahead and discretization in ilp*, in In Proceedings of the 7th International Workshop on Inductive Logic Programming, Springer-Verlag, 1997, pp. 77–85.
4. M. Botta and A. Giordana, *SMART+: A multi-strategy learning tool*, in IJCAI, 1993, pp. 937–945.
5. F. Divina and E. Marchiori, *Handling continuous attributes in an evolutionary inductive learner.*, IEEE Trans. Evolutionary Computation, 9 (2005), pp. 31–43.
6. D. Ienco, R. G. Pensa, and R. Meo, *Context-based distance learning for categorical data clustering*, in IDA, 2009, pp. 83–94.
7. N. Lavrac, S. Dzeroski, and M. Grobelnik, *Learning nonrecursive definitions of relations with linus*, in Proceedings of the European Working Session on Machine Learning, EWSL '91, London, UK, UK, 1991, Springer-Verlag, pp. 265–281.
8. S. Muggleton, M. Bain, J. Hayes-michie, and D. Michie, *An experimental comparison of human and machine learning formalisms*, in In Proceedings of the Sixth International Workshop on Machine Learning, Morgan Kaufmann, 1989, pp. 113–118.
9. O. Muñoz and R. MacKinney-Romero, *Multivalue in ILP*, in 20th International Conference on Inductive Logic Programming, 2011.
10. Srinivasan, Muggleton, A. Srinivasan, and S. H. Muggleton, *Mutagenesis: Ilp experiments in a non-determinate biological domain*, in Proceedings of the 4th International Workshop on Inductive Logic Programming, volume 237 of GMD-Studien, 1994, pp. 217–232.
11. A. Srinivasan and R. Camacho, *Experiments in numerical reasoning with inductive logic programming*, Journal of Logic Programming.
12. A. Srinivasan, R. D. King, S. Muggleton, and M. J. E. Sternberg, *Carcinogenesis predictions using ilp*, in ILP, 1997, pp. 273–287.