Induction of Abstraction Operators Using Unsupervised Discretization of Continuous Attributes

M. Biba, S. Ferilli, T.M.A. Basile, N. Di Mauro, and F. Esposito

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

Abstract. Most complex real-world domains are described by both relations among objects and numeric features. While ILP allows to handle relations, it is less suited to deal with numeric values. Abstraction operators can help in these cases by replacing specific values with intervals. Since it is difficult for humans to provide useful discretizations, we propose a method for automatically learning the abstraction operator from available observations. The resulting abstraction theories are then tested on a well-known dataset.

1 Introduction

Most complex real-world domains are described by numeric information, but attribute-value representation languages could be not enough powerful to capture their complexity, raising effectiveness problems for machine learning systems. In such domains, it is often necessary to adopt a more powerful representation language, like first-order logic that is able to describe relations between objects. However, while being very suitable for handling symbolic descriptions, it is less tailored to cope with numerical (and specifically real-valued) information. A possible solution to overcome such a problem, and enhance the learning process, is obtained by replacing specific numerical values with symbolic descriptors that represent the interval they belong to. Conceptually, this corresponds to the extension of pure induction with other inference strategies, and specifically with abstraction [21].

Abstraction deals with cases when learning can be more effective if it can take place at multiple (different) levels of complexity, which can be intended as a language bias shift. Indeed, it can be used by machine learning systems to perform a shift to a higher level language when the current one is not enough powerful to capture the target predicate definition (a well-known problem in the traditional approach to the inductive concept-learning). An Abstraction Theory (an operational representation of such a mapping) is used to perform such a shift to a higher level representation: It is a collection of alternative definitions for intermediate concepts.

In Machine Learning, a very useful kind of abstraction is discretization that corresponds to a co-domain reduction [21] where the domain of values taken by object descriptors, function or relation is reduced. Discretization, defined as a set of cutpoints over domains of attributes, represents an important preprocessing task for most data analysis tasks. In learning tasks, discretization has been proved to be very

useful. In [6] it is shown that discretization prior to induction can sometimes significantly improve the accuracy of an induction algorithm. In this work the performance of C4.5 did not degrade if data were discretized in advance and in two cases even improved significantly. In [2] the authors show that variable discretization increases the speed of the induction algorithms for very large datasets (as in common data mining tasks).

The handling of numeric attributes in ILP has been previously tackled by some other works. In [7] the numerical attributes are handled by transforming ILP problems to propositional form an thus use propositional learning approaches that have elaborate number-handling capabilities. Because of the limited expressiveness of propositional logic we want to deal in this paper with the numeric handling problem in a first order logic framework. In [13] and [1] numerical attributes are handled in ILP problems using supervised discretization and the results reported show that discretizaton increases efficiency at no cost of accuracy. The discretization method used in both works is a supervised one and it is based on the supervised discretization algorithm proposed in [9]. Supervised discretization methods require class information of objects whose properties are described by numerical values that are to be discretized. Thus, when such information is available these methods have shown good ability to produce significant discretization. However class information is not always available. Often many objects involved in a learning task have no class and in these cases unsupervised discretization methods could be useful. In this paper we propose an unsupervised discretization method to handle numerical information in an ILP problem. Another category of ILP systems such as Progol or Foil deal with numerical values by handling inequalities in clauses and generating numbers during the learning task. This method is quite expensive and a significant reduction of domain values into discrete intervals would be useful for efficiency concerns.

The paper is organized as follows. In Section 2 we describe the abstraction framework within which discretization is mapped as an abstraction operator. In Section 3 we describe a novel unsupervised discretization method that we use to automatically induce abstraction theories. In Section 4 we present experiments and in Section 5 we conclude.

2 Abstraction Framework

Abstraction is defined as a mapping between representations that are related to the same reference set but contain less detail (typically, only the information that is relevant to the achievement of the goal is maintained [22]. It is useful in inductive learning when the current language bias proves not to be expressive enough for representing concept descriptions that can explain the examples.

Definition 1. *Given two clausal theories T (ground theory) and T' (abstract theory) build upon different languages L and L' (and derivation rules), an abstraction is a triple (T, T', f) where f is a computable total mapping between clauses in L and those in L'*

An Abstraction Theory (an operational representation of *f*) is used to perform such a shift of language bias [17], [19] to a higher representation:

Definition 2. *An abstraction theory from L to L' is a consistent set of clauses* c:- $d_1,\ldots,d_m$ *where c is a literal built on predicates in L', and* d_j, $j = 1,\ldots,m$ *are literals built on predicates of L.*

i.e., it is a collection of intermediate concepts represented as a disjunction of alternative definitions. *Inverse resolution* [14] (inter-construction, intra-construction and absorption) can be a valuable mechanism to build and exploit abstraction theories as introduced in [11]. In this paper we are interested in the case of Datalog programs, where clauses are *flattened,* hence function-free.

Definition 3. (absorption) *Let C and D be two Datalog clauses. If* $\exists\ \theta$ *unifier such that* $S = body(D)\theta \subset body(C)$, *then applying the absorption operator yields the new clause C' such that* $head(C') = head(C)$ *and* $body(C) = (body(C) \setminus S) \cup \{head(D)\ \theta\}$

i.e., if all conditions in *D* are verified in the body of *C, the corresponding literals are eliminated and replaced by head*(D).

According to the framework proposed in [22], abstraction takes place by means of a set of operators, that generally includes operators for grouping indistinguishable objects into equivalence classes; grouping ground objects to form a compound object (that replaces them in the abstract world); merging values that are considered indistinguishable; reducing the arity of a function or relation (even up to the elimination of all arguments). Modifications are performed by mappings.

In this framework, discretization corresponds to the abstraction operator that merges values that are considered indistinguishable. Indeed, the aim of discretization is to represent a set of values through a reduced number of intervals. These should represent in a compact way the initial numerical domain without loss of information.

To be able to perform abstraction during the learning task, the system must be provided with an abstraction theory for the specific application domain (that according to Definitions 1 and 2) contains the operators encoding the abstraction mapping *f*, between languages *L* and *L'* represented as a set of clauses, i.e. domain rules. Usually such domain rules are hand-coded by the domain expert. However, very often providing proper abstraction theories, and specifically sensible discretizations of real values, is very difficult for humans, even domain experts. Under this motivation, after facing in previous work [10] the automatic learning of operators for grouping and ignoring objects, here we focus on the automatic inference of value-merging (i.e., discretization of continuous values into intervals) operators. These *shifting rules* will be applied in order to perform the shift in language bias according to the absorption operator presented in Definition 3.

3 Unsupervised Discretization

There are different axes by which discretization methods can be classified, according to the different directions followed by the implementation of discretization techniques due to different needs: global vs. local, splitting (top-down) vs. merging (bottom-up), direct vs. incremental and supervised vs. unsupervised.

Local methods, as exemplified by C4.5, discretize in a localized region of the instance space. (i.e. a subset of instances). On the other side, global methods use the entire instance space [4].

Splitting methods start with an empty list of cutpoints and, while splitting the intervals in a top-down fashion, produce progressively the cut-points that make up the discretization. On the contrary, merging methods start with all the possible cutpoints and, at each step of the discretization refinement, eliminate cut-points by merging intervals.

Direct methods divide the initial interval in *n* sub-intervals simultaneously (i.e., equal-width and equal-frequency), thus they need as a further input from the user the number of intervals to produce. Incremental methods [3] start with a simple discretization step and progressively improve the discretization, hence needing an additional criterion to stop the process.

Supervised discretization considers class information while unsupervised discretization does not. Equal-width and equal-frequency binning are simple techniques that perform unsupervised discretization without exploiting any class information. In these methods, continuous intervals are split into sub-intervals and it is up to the user specifying the width (range of values to include in a sub-interval) or frequency (number of instances in each sub-interval). These simple methods may not lead to good results when the continuous values are not compliant with the uniform distribution. Additionally, since outliers are not handled, they can produce results with low accuracy in the presence of skew data. Usually, to deal with these problems, class information has been used in supervised methods, but when no such information is available the only option is exploiting unsupervised methods. While there exist many supervised methods in literature, not much work has been done for synthesizing unsupervised methods. This could be due to the fact that discretization has been commonly associated with the classification task. Therefore, work on supervised methods is strongly motivated in those learning tasks where no class information is available. In particular, in many domains, learning algorithms deal only with discrete values. Among these learning settings, in many cases no class information can be exploited and unsupervised discretization methods such as simple binning are used.

The work presented in this section proposes a top-down, global, direct and unsupervised method for discretization. It exploits density estimation methods to select the cut-points during the discretization process. The number of cutpoints is computed by cross-validating the log-likelihood. We consider as candidate cutpoints those that fall between two instances of the attribute to be discretized. The space of all the possible cut-points to evaluate could grow for large datasets that have continuous attributes with many instances with different values among them. For this reason we developed and implemented an efficient algorithm of complexity $Nlog(N)$ where N is number of instances.

3.1 Non-parametric Density Estimation

Since data may be available under various distributions, it is not always straightforward to construct density functions from some given data. In parametric density estimation, an important assumption is made: available data has a density function that belongs to a known family of distributions, such as the normal distribution or the Gaussian one, having their own parameters for mean and variance. What a parametric method does is finding the values of these parameters that best fit the data. However, data may be complex and assumptions about the distributions that are forced upon the data may lead to models that do not fit well the data. In these cases, where making assumptions is difficult, non-parametric density functions are preferred.

Simple binning (histograms) is one of the most well-known non-parametric density methods. It consists in assigning the same value of the density function $\boldsymbol{f}$ to every instance that falls in the interval $[C - h/2, C + h/2)$, where C is the origin of the bin and h is the binwidth. The value of such a function is defined as follows (# stands for number of):

$$\boldsymbol{f} = \frac{1}{n * h} \#\{\text{instances that fall in} \left[C - \frac{h}{2}, C + \frac{h}{2} \right)\}$$

Once fixed the origin C of a bin, for every instance that falls in the interval centered in C and of width h, a block of size 1 by the bin width is placed over the interval (Figure 1).

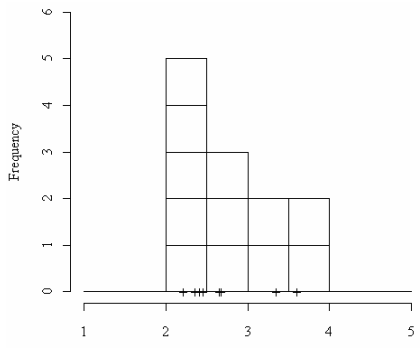

Figure 1. Simple binning places a block in every sub-interval for every instance x that falls in it

Here, it is important to note that, if one wants to get the density value in x, every other point in the same bin, contributes equally to the density in x, no matter how close or far away from x these points are. This is rather restricting because it does not give a real mirror of the data. In principle, points closer to x should be weighted more than other points that are far from it. The first step in doing this is eliminating the dependence on bin origins fixed a-priori and place the bin origins centered at every point x. Thus the following pseudo-formula:

$$\frac{1}{n * \text{binwidth}} \#\{\text{instances that fall in a bin } \underline{\text{containing}} \text{ x}\}$$

should be transformed in the following one:

$$\frac{1}{n * \text{binwidth}} \#\{\text{instances that fall in a bin } \underline{\text{around}} \text{ x}\}$$

The subtle but important difference in constructing binning density with the second formula, permits to place the bin around x and the calculation of the density is performed not in a bin containing x and depending from the origin C, but in a bin whose center is upon x. The bin center on x, allows successively to assign different weights to the other points in the same bin in terms of impact upon the density in x depending on the distance from x. If we consider intervals of width h centered on x, then the density function in x is given by the formula:

$$\boldsymbol{f} = \frac{1}{2hn} \#\{\text{instances that fall in } [\text{x} - h, \text{x} + h]\}$$

In this case, when constructing the density function, a box of width h is placed for every point that falls in the interval centered in x. These boxes (the dashed ones in Figure 2) are then added up, yielding the density function of Figure 2.

This provides a way for giving a more accurate view of what the density of the data is, called box kernel density estimate. However, the weights of the points that fall in the same bin as x have not been changed yet. In order to do this the kernel density function is introduced:

$$\boldsymbol{p} = \frac{1}{nh} \sum_{i=1}^{\text{n}} K\left(\frac{x - X_i}{h}\right)$$

where K is a weighting function. What this function does is providing a smart way of estimating the density in x, by counting the frequency of other points X_i in the same bin as x and weighting them differently depending on their distance from x.

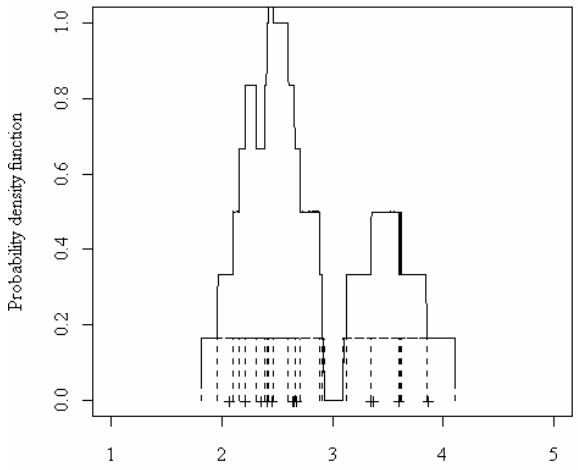

Figure 2. Placing a box for every instance in the interval around x and adding them up.

Contributions to the density value of $\boldsymbol{f}$ in x from points X_i vary, since those that are closer to x are weighted more than points that are further away. This property is fulfilled by many functions, that are called kernel functions. A kernel function K is usually a probability density functions that integrates to 1 and takes positive values in its domain. What is important for the density estimation does not reside in the kernel function itself (Gaussian, Epanechnikov or quadratic could be used) but in the bandwidth selection [18]. We will motivate our choice for the bandwidth (the value h in the case of kernel functions) selection problem in the next section where we introduce the problem of cutting intervals based on the density induced by the cut and the density given by the above kernel density estimation.

3.2 The Scoring Function for the Cutpoints

The aim of discretization is always to produce sub-intervals whose induced density over the instances best fits the available data. The first problem to be solved is where to cut. While most supervised top-down discretization method cut exactly at the points in the main interval to discretize that represent instances of the data, we decided to cut in the middle points between instance values. The advantage is that this cutting strategy avoids the need of deciding whether the point at which the cut is performed is to be included in the left or in the right sub-interval.
The second question is which (sub-)interval should be cut/split next among those produced at a given step of the discretization process. Such a choice must be driven by the objective of capturing the significant changes of density in different separated bins. Our proposal is to evaluate all the possible cut-points in all the sub-intervals, by

assigning to each of them a score according to a method whose meaning is as follows. Given a single interval to split, any of its cut-points produces two bins and thus induces upon the initial interval two densities, computed using the simple binning density estimation formula. Such a formula, as shown in the previous section, assigns the same density value of the function ***f*** to every instance in the bin and ignores the distance from x of the other instances of the bin when computing the density in x. Every sub-interval produced has an averaged binned density (the binned density in each point) that is different from the density estimated with the kernel function. The less this difference is, the more the sub-interval fits the data well, i.e. the better this binning is, and hence there is no reason to split it. On the contrary, the idea underlying our discretization algorithm is that, when splitting, one must search for the next two worst sub-intervals to produce, where "worst" means that the density shown by each of the sub-intervals is much different than it would be if the distances among points in the intervals and a weighting function were considered. The identified worst sub-intervals are just those to be split to produce other intervals, because they do not fit the data well. In this way intervals whose density differs much from the real data situation are eliminated, and replaced by other sub-intervals. In order to achieve the density computed by the kernel density function we should reproduce a splitting of the main interval such as that in Figure 2.

An obvious question that arises is: when a given sub-interval is not to be cut anymore? Indeed, searching for the worst sub-intervals, there are always good candidates to be split. This is true, but on the other hand at each step of the algorithms we can split only one sub-intervals in other two. Thus if there are more than one sub-interval (this is the case after the first split) to be split, the scoring function of the cut-points allows to choose the sub-interval to split.

At each step of the discretization process, we must choose from different sub-intervals to split. In every sub-interval we identify as candidate cut-points all the middle points between the instances. For each of the candidate cut-points T we compute a score as follows:

$$\text{Score(T)} = \sum_{i=1}^{k}(p(x_i) - f(x_i)) + \sum_{i=k+1}^{n}(p(x_i) - f(x_i))$$

where $i = 1,..,k$ refers to the instances that fall into the left sub-interval and $i = k+1,..,n$ to the instances that fall into the right bin. The density functions p and f are respectively the kernel density function and the simple binning density function. These functions are computed as follows:

$$f(x_i) = \frac{m}{w * N}$$

where m is the number of instances that fall in the (left or right) bin, w is the binwidth and N is the number of instances in the interval that is being split. The kernel density estimator is given by the formula:

$$p(x_i) = \frac{1}{hN}\sum_{j=1}^{N} K\left(\frac{x_i - X_j}{h}\right)$$

where h is the bandwidth and K is a kernel function. In this framework for discretization, it still remains to be clarified how the bandwidth of the kernel density estimator is chosen. Although there are several ways to do it, as reported in [18], in fact in this context we are not interested in the density computed by a classic kernel density estimator that considers globally the entire set of available instances. The classic way a kernel density estimation works considers N as the total number of instances in the initial interval and chooses h as the smoothing parameter. The choice of h is not easy and various techniques have been investigated to find an optimal h. Our proposal, in this context, is to adapt the classic kernel density estimator by taking h equal to the binwidth w, specified as follows. Indeed, as can be seen from the formula of $p(x_i)$, instances that are more distant than h from x_i, contribute with weight equal to zero to the density of x_i. Hence, if a sub-interval (bin) under consideration has binwidth h, only the instances that fall in it will contribute, depending on their distance from x_i, to the density in x_i. As we are interested in knowing how the current binned density (induced by the candidate cut-point and computed by f with binwidth w) differs from the density in the same bin but computed weighting the contributions of X_j to the density in x_i on the basis of the distance $x_i - X_j$, it is useless to consider, for the function p, a bandwidth greater than w.

3.3 The Discretization Algorithm

Once a scoring function has been synthesized, we explain how the discretization algorithm works. It starts with an empty list of cut-points (that can be implemented as a priority queue in order to maintain, at each step, the cut-points ordered after their value according to the scoring function) and another priority queue that contains the sub-intervals generated thus far. Let us see it through an example. Suppose the initial interval to be discretized is the one in Figure 3 (frequencies of the instances are not shown).

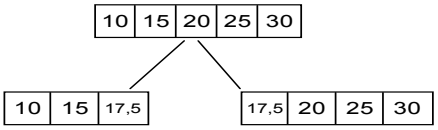

Figure 3. The first cut

The candidate cut-points are placed in the middle of adjacent instances: 12.5, 17.5, 22.5, 27.5; the sub-intervals produced by cut-point 12.5 are [10 , 12.5] and [12.5 , 30], and similarly for all the other cut-points. Now, suppose that, computing the scoring function for each cut-point, the greatest value (indicating the cut-point that produces the next two worst sub-intervals) is reached by the cut-point 17.5. Then the sub-

intervals are: [10 , 17.5] and [17.5 , 30] and the list of candidate cut-points becomes <12.5, 16.25, 18.75, 22.5, 27.5>. Suppose the scoring function evaluates as follows: Score(12.5) = 40, Score(16.25) = 22, Score(18.75) = 11, Score(22.5) = 51, Score(27.5) = 28. The algorithm selects 22.5 as the best cut-point and splits the corresponding interval as shown in Figure 4.

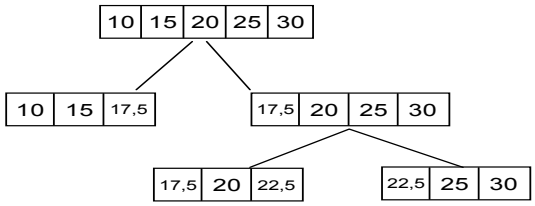

Figure 4. The second cut

This second cut produces two new sub-intervals and hence the current discretization is made up of three sub-intervals: [10 , 17.5], [17.5 , 22.5], [22.5 , 30], with candidate cut-points <12.5, 18.75, 23.75, 27,75>. Suppose values of the scoring function are as follows: Score(12.5) = 40, Score(18.75) = 20, Score(23.75) = 35, Score(27,5) = 48. The best cut-point 27.5 suggests the third cut and the discretization becomes [10 , 17.5], [17.5 , 22.5], [22.5 , 27.5], [27.5 , 30]. Thus, the algorithm refines those sub-intervals that show worst fit to the data. A note is worth: in some cases it might happen that a split is performed even if one of the two sub-intervals it produces (which could be the left or the right one) shows such a good fit, compared to the other sub-intervals, that it is not split in the future. This is not strange, since the scoring function evaluates the overall fit of the two sub-intervals. This is the case of the first cut in the present example: the cut-point 17.5 has been chosen, where the left sub-interval [10 , 17.5] shows good fit to the data in terms of density while the right one [17.5 , 30] shows bad fit. In this case the interval [10 , 17.5] will not be cut before the interval [17.5 , 30] and perhaps will remain untouched till the end of the discretization algorithm. The algorithm will stop cutting when the stopping criterion (the maximal number of cut-points, computed by a procedure explained in the next paragraph) is met.

3.4 Stopping Criteria and Complexity

The definition of a stopping criterion is fundamental, to prevent the algorithm from continuing to cut until each bin contains a single instance. Even without reaching such an extreme situation, the risk of running into overfitting the model is real, because, as usual in the literature, we use log-likelihood to evaluate the density estimators, the simple binning and the kernel density estimate. As a solution, instead of requiring a specific number of intervals (that could be too rigid and not based on valid assumptions), we propose the use of cross-validation to provide an unbiased estimation of how the model fits the real distribution. For the experiments performed the 10-fold cross-validation was used. For each fold the algorithm computes the

stopping criterion as follows: Supposing there are $N - 1$ candidate cut-points, for each of them the cross-validated log-likelihood is computed. In order to optimize performance, at each step a structure maintains the sub-intervals in the current discretization and the corresponding splitting values, so that only the new values for the interval to be split have to be computed at each step. Thus the algorithm that computes the log-likelihood for the N – 1 cut-points is performed 10 times overall. The number of cut-points that shows the maximum value of the averaged log-likelihood on the test folds is chosen as the best. The log-likelihood on the test data is given by the following formula:

$$\text{Log-likelihood} = \sum_{j=1}^{n} n_{j-test} \log \frac{n_{j-train}}{w * N}$$

where $n_{j\text{-}train}$ is the number of training instances in bin j, $n_{j\text{-}test}$ is the number of test instances that fall in bin j, N is the total number of instances and w is the width of bin j. As regards the kernel density estimator complexity, from the formula of p, it can be deduced that the complexity for evaluating the kernel density in N points is N^2. For univariate data, the complexity problem has been solved by the algorithms proposed in [12] and [20] which compute the kernel density estimate in $O(N+N)$ instead of $O(N^2)$. In our context we deal only with univariate data because only single continuous attributes have to processed, and thus for N instances, the theoretical complexity of the algorithm is $O(N\log N)$.

4 Experiments

In this section we report some results on experiments with the proposed discretization method compared to other discretization methods. Then we use our method to generate abstraction theories and test them in two ILP systems for the mutagenesis task.

4.1 Experiments for the Discretization Method

In order to assess the validity and performance of the proposed discretization algorithm, we have performed experiments on several datasets taken from the UCI repository and classically used in the literature to evaluate discretization algorithms in the past. Specifically, the dataset used are: autos, bupa, wine, ionosphere, ecoli, sonar, glass, heart, hepatitis, arrhythmia, anneal, cylinder, and auto-mpg. These datasets contain a large set of numeric attributes of various types, from which 200 continuous attributes were extracted at random and used to test the discretization algorithm.

Since our goal here (for efficiency concern in ILP) is to produce compact representations of continuous attributes through discrete intervals, we want to have a number of intervals as small as possible. For this reason, from empirical results we decided that in order to have a reduced number of intervals that could help the efficiency of the learning ILP task, in the experiments we report here, there are not allowed cut points that produce sub-intervals with a number of instances lower than

5% of the total number of instances. This trick, although constraining, helped in producing less intervals without degrading the overall results of the discretization process.

In order to evaluate the discretization carried out by the proposed algorithm with respect to other algorithms in the literature, we compared it to three other methods: equal-width with fixed number of bins (we use 10 for the experiments), equal-frequency with fixed number of bins (we use 10 for the experiments), equal-width cross-validated for the number of bins. The comparison was made along the log-likelihood on the test data using a 10-fold cross-validation methodology. The results on the test folds were compared through a paired t-test as regards cross-validated log-likelihood. Table 1 presents the results of the t-test based on cross-validated log-likelihood with a risk level $\alpha = 0.05$. It shows the number of continuous attributes whose discretization through our method was significantly better, equal or significantly worst compared to the other methods.

	Our method significantly more accurate	Not significantly different	Our method significantly less accurate
Equal-Width (10 bins)	59	132	9
Equal-Frequency (10 bins)	56	136	8
Equal-Width Cross-validated	44	150	6

Table 1. Cross-validated log-likelihood t-test comparison with other discretization methods

4.2 Experiments in ILP

Given the good performance of the proposed discretization procedure, we decided to exploit it as a tool for the automatic induction of a value-merging abstraction operator to be used in abstraction theories for supporting ILP systems in dealing with numerical information. Specifically, we embedded it in the symbolic version of the ILP system INTHELEX [8] and ran experiments on the classical mutagenesis dataset [5] from which we considered the atom bond descriptions, the indicator variables, *logp* and *lumo*. The aim was studying the system performance when provided with the automatically learned abstraction operator concerning the three continuous attributes that describe the examples, and specifically: *logp*, *lumo* and *charge*.

As measures of performance, we use predictive accuracy, runtime and theory complexity intended as number of clauses. Initially we tested the abstraction theory automatically generated by the discretization algorithm by letting it choose the best number of intervals, which was 19, 18 and 16 for logp, lumo and charge, respectively. A 10-fold cross-validation produced these results: 6985 seconds as processor time, 11 clauses and 83.5% predictive accuracy. In order to understand how the learning task is influenced by the number of intervals, we generated other abstraction theories by forcing the number of intervals in the discretization method. Among different numbers of intervals we found that the best results in terms of predictive accuracy (86.9%) were reached by generating 7 intervals for each of the three continuous attributes, at the expense of a slightly more complex theory: 4258 seconds of

processor time and 12 clauses. This is quite interesting because it shows that by searching in the space of abstraction theories for a theory that can focus on more relevant information, it is possible to increase the performance of the learning system. Although the automatically found triple 19-18-16 produced good results compared to the maximal performance obtained with the triple 7-7-7, it is useful to investigate how the best number of intervals can be achieved.

A natural question that arises is: what is the performance of an ILP system that uses discretization to handle numerical attributes towards one that does not perform discretization such as Progol [15]. For this reason, we decided to experiment the discretization method with Progol by performing the discretization on the mutagenesis dataset before the learning task of Progol starts. This way we can analyse how discretization affects the learning process in ILP. Moreover, we can empirically show how unsupervised discretization can help in handling numerical attributes. Table 2 shows the results of the experiments conducted with CProgol 4.4 [16] by using the discretization obtained with the triple 7-7-7 for the continuous attributes in mutagenesis.

	Time	Accuracy
CProgol and normal dataset	1255,95 sec	83.49 %
CProgol and discretized dataset	978,10 sec	84,11 %

Table 2. Experiments with CProgol 4.4.

We can see from the table that efficiency of Progol improves when learning is performed on a discretized dataset at no significant changes in accuracy. This means that for large datasets in mining tasks, discretization can help to achieve great improvements in efficiency.

5 Discussion and Future Work

The experiments with the ILP system INTHELEX show that identifying the best abstraction theory upon numerical attributes leads to improvements in accuracy and efficiency for the learning task. Since it is not easy to provide manually this theory, the unsupervised method we have proposed can be useful to handle numerical attributes in an ILP context.

The experiments with Progol show that learning in ILP can benefit from discretization, and in particular from unsupervised discretization. The gain in efficiency by discretization can be exploited for hard tasks of ILP like those of mining very large relational datasets that contain numerical attributes and where propositional methods (that can handle numerical attributes) cannot be used because of their limits in handling this kind of datasets.

As future work, we plan to do more experiments on larger relational datasets that contain numerical attributes. Improvements to the discretization method can also be done in order to automatically find the best number of intervals that leads to the best abstraction theory.

References

[1] H. Blockeel, L. De Raedt: Lookahead and Discretization in ILP. In N. Lavrac, and S. Dzeroski, editors. *Inductive Logic Programming: Proceedings of the Seventh International Workshop*, LNAI, volume 1297, pages 77-84, Springer, Berlin, 1997.

[2] J. Cattlet. On changing continuous attributes into ordered discrete attributes. In Y. Kodretoff, ed., *Proceedings of the European Working Session on Learning",* Berlin, Germany: Springer-Verlag, pages 164-178, 1991.

[3] J. Cerquides, R.L. Mantaras Proposal and empirical comparison of a parallelizable distance-based discretization method. In *KDD97. Third International Conference on Knowledge Discovery and Data Mining*, pp. 139 – 142, 1997.

[4] M.R. Chmielevski, and J.W. Grzymala-Busse. Global discretization of continuous attributes on preprocessing for machine learning. In *Third International Workshop on Rough Sets and Soft Computing*, pp. 294-301, 1994.

[5] A.K. Debnath, R.L Lopez de Compadre, G. Debnath, A.J. Schusterman, and C. Hansch. Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal chemistry,* 34(2): 786 – 797, 1991.

[6] J. Dougherty, R. Kohavi, and M. Salami. Supervised and unsupervised discretization of continuous features. In A. Prieditis and S. Russell, editors, *Proc.Twelfth International Conference on Machine Learning,* Morgan Kaufmann, 1995.

[7] S. Dzeroski, L. Todorovski, and T. Urbancic. Handling real numbers in ILP: A steptowards successful behavioural cloning. In *Proc. Eighth European Conference on Machine Learning*, pages 283-286. Springer, Berlin, 1995.

[8] F. Esposito, S. Ferilli, N. Fanizzi, T. M.A. Basile, and N. Di Mauro. Incremental Multistrategy Learning for Document Processing. *Applied Artificial Intelligence: An Internationa Journal*, 17(8/9):859–883, Taylor \& Francis, 2003.

[9] U. M Fayyad, K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, pages 1022 – 1027, 1993.

[10] S. Ferilli, T.M.A. Basile, N. Di Mauro, and F. Esposito. Automatic Induction of Abduction and Abstraction Theories from Observations. In S. Kramer and B. Pfahringer, editors, *Inductive Logic Programming*, volume 3625, LNAI, pp. 103–120, Springer Verlag, 2005

[11] A. Giordana, D. Roverso, and L. Saitta. Abstracting concepts with inverse resolution. In *Proceedings of the 8th International Workshop on Machine Learning,* pages 142-146, Evanston, IL, Morgan Kaufmann, 1991.

[12] L. Greengard, J. Strain. The fast Gauss Transform. *SIAM Journal of Scientific and statistical computing*. 12, 1, 79-94, 1991.

[13] W. van Laer, L. De Raedt, and S. Dzeroski. On multi-class problems and discretization in inductive logic programming. In *Proc. Tenth International Symposium on Foundations of Intelligent Systems*, pages 277-286. Springer, Berlin, 1997.

[14] S. H. Mugletton and L. De Raedt. Inductive Logic Programming . *Journal of Logic Programming: Theory and Methods, 19: 629-679, 1994.*

[15] S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245-286,

1995.
[16] S.H. Muggleton and J. Firth. CProgol4.4: a tutorial introduction. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*, pages 160-188. Springer-Verlag, 2001.
[17] L. De Raedt. Interactive Theory Revision – An Inductive Logic Programming Approach, Accademic Press, 1992.
[18] B.W. Silverman. Density estimation for statistics and data analysis. *Chapman and Hall*, London, 1986.
[19] P.E. Utgoff. Shift of bias for inductive concept learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning, an artificial intelligence approach,* volume II, pages 107 – 148. Morgan Kaufmann, Los Altos, CA, 1986.
[20] C.Yang, R. Duraiswami, and N. Gumerov. Improved fast Gauss transform. *Tech. Rep.CS-TR-4495*, Dept. of Computer Science, University of Maryland, College Park. 2003.
[21] J. D. Zucker. A grounded theory of abstraction in artificial intelligence. In *Philosophical Transactions of the Royal Society B: Biological Sciences volume 358, pages 1293 – 1309, 2003.*
[22] J.-D. Zucker. Semantic abstraction for concept representation and learning. In R. S. Michalski and L. Saitta, editors, *Proceedings of the 4th International Workshop on Multistrategy Learning*, pages 157–164, 1998.