Learning Disjunctive Normal Forms in a Dual Classifier System (Extended Abstract)

Cathy Escazut Philippe Collard

University of Nice-Sophia Antipolis-CNRS, Laboratory I3S, Bât. 4
250 Av. Albert Einstein, Sophia Antipolis, 06560 Valbonne — FRANCE
{escazut,pc}@unice.fr

Abstract. Genetics-Based Machine Learning systems suffer from many problems as representational weaknesses. We propose to introduce more general structures we used to learn disjunctive normal forms. Results show how our model can be used to discover and maintain complete classifier solutions.

1 Genetics-Based Machine Learning

Genetics-Based Machine Learning systems use *Genetic Algorithms* (GAs) as discovery heuristic. Such algorithms modify a population of potential solutions, using operators stemmed on natural genetics: reproduction, crossover, mutation. *Learning Classifier System* (LCS) implementations have shown the potential of this paradigm for machine learning, and also some limitations. LCSs automatically discover rules to perform tasks [2]. Each classifier is an "if-then" rule, with a condition part and an action part. New classifiers are generated by genetic operators applied to existing rules. Each condition part of a classifier, also called *schema* or *hyperplane*, is a string of length λ over the alphabet {0,1,#}, where # is a wildcard character. Classifier representations appear to be a simple, effective method for implementing computational systems. However, many subtleties arise within the representation [5]. Indeed, disjunctions are hardly represented with a single rule. Consequently, the solution set and the search effort are increased.

2 Relational Schemata

In order to allow a more natural expression of solutions we propose to consider not only the value on each locus but also the relations (equality and inequality) between the values on different locus. We thus define *relational schemata*, called *R–schema*, as a string built over the alphabet {X,X',#}[1]. As standard schema only express values on the different locus, we call them *P–schemata* or *positional* schemata. We are now going to study R–schemata in relation to properties Radcliffe [4] thinks requisite for a useful representation. Our aim is not to show R–schemata are better than P–schemata but to show they are complementary.

[1] The two symbols X and X' represent two complementary variables.

The closure: *The intersection of any pair of compatible*[2] *schemata should itself be a schema.* This property allows search to be gradually refined. Obviously, P–schemata and corroborating[3] R–schemata possess the closure property. Whereas the intersection of two non-corroborating R–schema is not a R–schema. In this sense, we say that R–schemata are semi-closed for intersection.

The respect: *Crossing two members of any schema should produce another member of that schema.* This property is necessary to keep good schemata. Crossover operators respect P–schemata but not R–schemata.

The proper assortment: *Given instances of two compatible schemata, it should be possible to cross them to produce a child which is an instance of both schemata.* This property allows the recombination of usefull schemata. Only uniform crossovers properly assort P–schemata, but not R–schemata.

Using R–schemata, we are able to represent more hyperplanes. For instance, the R–schema XX represents the subset {00,11}. But all the hyperplanes, for instance {011,000}, are not representable. We are thus going to extend the expressiveness of a schema defining a RP–schema as a string built over the alphabet {0,1,X,X',#}. So, a RP–schema can be obtained by the intersection of a R–schema and a P–schema. For instance, the intersection of 0## and #XX is the RP–schema 0XX.R–schemata don't possess the requisite properties and the use of variables increases the size of the space. That is why R–schemata are not explicitly used in LCS. Thus, a solution is to implement the notion of R–schema in an implicit way.

3 Implementation of Relational Schemata

The aim of a implicit implementation of R–schemata is to keep the alphabet {0,1,#} unchanged and to have the properties possessed. A string, in our system, will consist of two parts: the first one, a single bit called *head-bit*, contains the information needed for understanding the rest of the string. More formally, let the search space be $\Omega=\{0,1\}^{\lambda}$, and the *dual space* be $<\Omega>=\{0,1\}\mathrm{x}\Omega$. We define a *transcription* function, τ, from the dual space $<\Omega>$ to the basic space Ω by $\forall\omega \in \Omega \quad \tau(0\omega) = \omega$ *and* $\tau(1\omega) = \omega'$ where ω and ω' are complementary strings. It is worth noticing that different strings may be interpreted in the same way. For instance the two conditions $\underline{0}\,0101$ and $\underline{1}\,1010$ are both decoded as 0101. We call them *dual* strings. Moreover, when the head-bit is undetermined the use of a variable is requisite. For instance, the P–schema $\underline{\#}\,01\#0$ becomes the R–schema XX'#X. Does implicit R–schemata possess all the properties stated previously?

The respect: A GA applied on $<\Omega>$, through a choice between dual strings, allows R–schemata of Ω to be respected. For example, the set of the members of R–schema X#X' is not closed, but a corresponding one in $<\Omega>$, for instance $\{\underline{0}\,001,\underline{0}\,011,\underline{1}\,011,\underline{1}\,001\}$, is the P–schema $\underline{\#}\,0\#1$ possessing the respect property.

[2] Two schemata are *compatible* if there exists a string being a member of both.

[3] Two compatible R–schemata are *corroborating* if they share at least one locus with variable.

The proper assortment: A GA applied on $<\Omega>$ through a choice between dual strings, allows a uniform crossovers to properly assort R–schemata. For example, let us consider the compatible R–schemata X#X' and #XX and be 001 and 000 two members. If we represent them in $<\Omega>$ by respectively $\underline{0}001$ and $\underline{1}111$, a uniform crossover breeds $\underline{0}011$ in the intersection XX'X'.

The ergodicity: *It should be possible through a finite sequence of application of genetic operators, to access any point in the search space.* This property is the "raison d'être" of the mutation. For two λ-strings of Ω, at least λ mutations are needed to reach a point, while in $<\Omega>$, this number is smaller than $int(\frac{\lambda}{2}) + 1$.

A RP–schema is the intersection of a P–schema and a R–schema. So it can be represented by a conjunction $[a, b]$ where a is a P–schema of $\Omega(\lambda)$ and b a P–schema of $\Omega(\lambda+1)$[4]. The condition is satisfied when the two fields are satisfied [3]. Another way to implement RP–schemata, is to use a *mask*: the condition is composed with a mask (λ bits), and with a P–schema of $\Omega(\lambda+1)$. Only the loci of the P–schema corresponding to a 1 in the mask will be influenced by the head bit. For instance the RP–schema 0XX can be implemented by [011, $\underline{\#}000$].

4 Learning disjunctive normal forms

To validate our proposition, we applied the algorithm on a similarity-based learning problem: the LCS has to learn a concise disjunctive normal form [6]. Each rule represents a conjunction of attributes. A set of classifiers represents a disjunction of such conjunctions. We test our system on two well-known problems: the XOR problem and the multiplexer function. The LCS used, is an adaptation of the one described by D.E. Goldberg [2]. The results obtained do not represent our "best efforts"; rather, they are intended as comparative results.

The XOR Problem

Schemata in the XOR problem are highly epistatic. LCSs have difficulties in learning such problems. In both cases (dual and basic approach), the initial population is randomly created with 22 classifiers. Results show that our system behaves in a better way than a standard LCS: approximately beyond cycle 2000, the solution set is found out: the final population contains 2 dominating classifiers ($\underline{\#}00$:0 and $\underline{\#}01$:1). The undetermined head-bit leads to the use of variables (XX:0 and XX':1) and so the use of less specialized classifiers. While the standard LCSs needs 5000 cycles to discover the four totally specialized classifiers: 00:0, 01:1, 10:1 and 11:0[5]. The dual learning is realized at an upper level of abstraction.

The Multiplexer Function

Two multiplexer functions, like the one described in [2], have been tested with different goals. The first one was to make good RP-schemata to appear in the population and the second one as to improve the success rate. In both trials, two fields compose the condition of the classifiers: the rightmost bits are for the

[4] Nervertheless, the head-bit of b must be undetermined and the schema a must be undetermined on the specified locus of b.

[5] The system has no other possibility than learning the solution by heart.

address lines, and the remaining ones are for the data. A head-bit is added in the dual approach. The action part represents the system's answer.

In the 6-line multiplexer, 2 bits are for the address and 4 for the data. The RP–schemata are expressed using a mask. For instance, the rule X###11:X will be represented by the string: [111100, $\underline{\#}$0###11]:0. Moreover, in this case, the head-bit has to influence the action in order to make variables emerged. The initial population is randomly created with 100 classifiers. The GA is invoked every 2500 iterations, and 20% of the current population undergoes genetic operators. After 50,000 iterations the minimal solution set[6] dominates into the dual population, while in the standard one 8 rules are needed. Once more, the use of R–schemata allowed to learn at an upper level of abstraction.

We also test our system on an 11-line multiplexer (3 lines for the address and 8 for the data), in order to improve the success rate of the LCS. The initial population is randomly created with 300 classifiers. The GA is invoked every 5000 iterations, and 20% of the current population undergoes genetic operators. The performance average of the two systems increases from the beginning, but the one of our LCS evolves faster. Indeed, since the cycle 35,000 the average is above 80%, against 71% for standard LCS. At the end, it reaches 83% with our system, while it is only near 72% with standard LCS.

5 Conclusion

This paper has presented a new LCS based on general structures called *relational schemata*. To each string is associated its bitwise complement. So we introduce in the population dual strings having the same meaning. We have shown how minimal classifier sets can be found using implicit relationl schemata. Further work is aimed at applying the dual LCS to more difficult tasks.

References

1. P. Collard, J.P. Aurand. DGA: An efficient genetic algorithm. In *ECAI'94: European Conference on Artificial Intelligence*, 1994.
2. D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Reading, MA : Addison-Wesley, 1989.
3. J. H. Holland. Escaping brittleness : The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In *Machine Learning II.* Morgan Kaufmann, 1986.
4. N. J. Radcliffe. Forma analysis and random respectful recombination. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA, 1991. Morgan Kaufmann.
5. D. Schuurmans, J. Schaeffer. Representational difficulties with classifier systems. In *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.
6. S. W. Wilson. Classifier systems and the animat problem. *Machine Learning*, 2(3),1987.

[6] {X### 11:X, #X## 10:X, ##X# 01:X, ###X 00:X}.