Learning from Recursive, Tree Structured Examples

P. Jappy, M.C. Daniel-Vatonne, O. Gascuel, and C. de la Higuera
DIF - LIRMM, 161 rue Ada, 34392 - Montpellier - FRANCE

Abstract In this paper, we propose an example representation system that combines a greater expressive richness than that of the Boolean framework and an analogous treatment complexity. The model we have chosen is algebraic, and has been used up to now to cope with program semantics [4]. The examples are represented by labelled, recursive typed trees. A signature enables us to define the set of all allowed (partial or complete) representations. This model properly contains Boolean representations. We show that in the PAC framework defined by Valiant [10], the extensions to this model of two Boolean formula classes: k-DNF and k-DL, remain polynomially learnable.

1 Introduction

This paper deals with data representation, in Inductive Learning. Two main trends are found in the literature:

The first and older one uses an attribute-values type language. When the attribute values are symbolic, this type of language is very close to propositional logic and to the Boolean framework. Learning from these attribute-values representations is "often" "quite" easy. This was shown from a theoretical point of view within the COLT. For instance k-DNF, k-CNF [10] and k-DL [9] are three Boolean formula classes that are polynomially learnable as defined in [10]. Practically, the successful applications of the attribute-values representation (in Machine Learning, but in Statistics and Pattern Recognition also) are extremely numerous. Of course, and that is the tradeback for efficiency, this type of representation is relatively poor.

The second trend tends to use a "structural" representation close to predicate logic or to semantic networks. For the main part, this approach comes from Artificial Intelligence and in particular the work of Winston [11]. This type of representation is richer than the previous one. Not only can we give the global characteristics of the example (e.g. *the colour, the shape...*), but also the relations that link its components (e.g. it is composed of a sphere *on top of* a cube...). The drawback is the algorithmic complexity problems inherent in this type of representation. This was shown by Haussler [6] within the framework of the COLT. However, recent work in Inductive Logic Progamming [3,8] has produced positive results on some very restricted FOL classes.

This paper proposes an "in between" way. We try to combine a complexity similar to that of the Boolean framework with a greater expressive power. The model we have chosen to represent the data has been used so far in algebraic program semantics [4]. The examples are represented by labelled, recursive typed trees called typed terms. A signature enables us to define the set of all allowed representations (be they partial or complete). The idea of using trees stems from the fact that they are close to the "limit of polynomiality". Forest matching, for instance, is already NP-complete. The fact that we dispose of a generation mechanism, rather than simple tree-like representations, enables us to better comprehend the language and its properties. For example, it allows us to define, and eventually generate, all specialisations of a given description. This representation model and its relation to the Boolean one are sketched in Section 2. Our goal is to dispose of a

representation system that is richer than the Boolean one, yet remains of similar algorithmic complexity. Proving that this is the case cannot be done out of a precise context. This is why we focus on PAC learning and show (in Section 3) that the natural extensions of two of the main Boolean formula classes, k-DNF and k-DL, stay polynomially learnable.

2 Typed Terms

Typed terms are built using an algebraic structure called a *signature* which determines the set of authorised representations. This contains the allowed symbols along with their types and argument types. With such a structure, terms are constructed recursively in a similar manner to mathematical functions. This endows them with a tree structure having good complexity properties and a natural intuitive graphic representation.

A signature is a quadruple (S, F, σ, α) where :
- S is a finite set of types (also called *sorts*).
- F is a finite set of symbols.
- $\sigma : F \text{ -----> } S$ is a many-to-one mapping. For any ϕ in F, $\sigma(\phi)$ is called type of ϕ.
- $\alpha : F \text{ -----> } S^*$ is a mapping (where S^* is the free monoid generated by S). For any symbol ϕ in F , $\alpha(\phi)$ gives the order and type of ϕ's arguments, with $\alpha(\phi) = \varepsilon$ the empty word when ϕ has no arguments. $/\alpha(\phi)/$ the length of $\alpha(\phi)$, is called ϕ's arity.

Furthermore, to each type s, is implicitly added a special symbol Ω_S meaning "unknown" such that $\alpha(\Omega_S) = \varepsilon$. Terms containing one or more of these symbols are partial descriptions of objects and the others are said to be completely specified. In the following, we assume that all the examples used for learning are completely specified.

The set of terms of type s (noted T_S) is the smallest set such that:

$$\forall \phi \in F, \sigma(\phi) = s \text{ and } \alpha(\phi) = \varepsilon \Rightarrow \phi \in T_S \quad \text{and}$$

$$\forall \phi \in F, \sigma(\phi) = s, \alpha(\phi) = s_1 \ldots s_n \text{ and } \forall i \in [1,n]\ t_i \in T_{S_i} \Rightarrow \phi(t_1, \ldots, t_n) \in T_S$$

Note that this definition is recursive and so allows direct implementation.

The generalisation relation between terms is defined by : t is more general than t' (noted $t \leq t'$) iff t is Ω or both terms have the same root ϕ and all arguments of ϕ in t are more general than those in t'. Relation $\leq$ is a partial order of minimal element Ω_S. The generalisation test $t \leq t'$ is linear in the size of t.

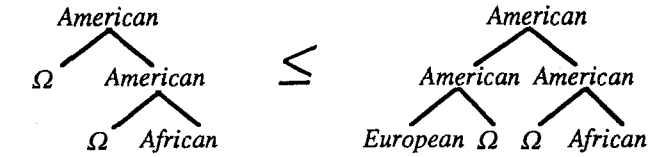
The size of a term is the sum of all its non Ω symbols. This size is a measure of the information content of the term. This allows us to compute N_k, the number of terms of a given type s and of size at most k, which is needed for learnability results. This value is :

$$N_k \leq k!\, a^k f^k \qquad \text{where } a = max\{/\alpha(\phi)/, \phi \in F\} \text{ and } f = /F/$$

Example: The following signature can be used to describe the origins of present day Americans, tracing through the genealogy of a person until it reaches one of the following possibilities: a Native American ancestor, or an immigrant from one of the continents.

$S = \{r\}$ where r stands for race.

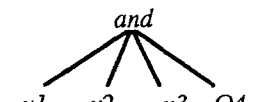
F	σ	α	F	σ	α
American	r	rr	*African*	r	ε
Asian	r	ε	*European*	r	ε
Hispanic	r	ε	*Native*	r	ε



Remark: This simplified signature admits only one type. The need for multisortedness appears when one wishes to add extra characteristics. Furthermore, the order of the arguments isn't commutative and in this example the left son of a node describes (by convention) the ancestors on the father's side and the right son those on the mother's.

Typed terms and Boolean representations

Any Boolean term can be simulated by a typed term using the following signature: each variable V_i is assigned a type and two literals (v_i and $\neg v_i$) plus the corresponding Ω, all having the same common root. So any literal can be true, false or unknown, the root representing their conjunction. This shows that our model generalises the Boolean one.



F	α	σ
and	$V_1V_2V_3V_4$	*conj*
v_1	ε	V_1
$\neg v_1$	ε	V_1
v_2	ε	$V_2\ldots$

It follows that the extension of non learnable Boolean function classes to typed terms will not be polynomially learnable either. This argument is in no way incompatible with the one stating that the superset of a non learnable class can be learnable : we simply translate a Boolean learning problem into another in the typed term framework and get our result through reduction. So, we will only concentrate on two learnable Boolean classes, namely k-DNF and k-DL. We define their natural extensions by replacing Boolean terms by their typed counterparts, and note these new classes k-TDNF and k-TDL respectively (where k now represents the maximum size of terms rather than the number of literals). In doing so, we have to extend Valiant's definition of polynomial learnability [10] by replacing the complexity parameter n (the number of Boolean variables) by a, the maximum arity of a symbol and f the number of symbols in the signature. Note that this extrapolation is consistent since using the above transformation, we have $a = n$ and $f = 2n+1$.

3 Polynomial learnability of k - TDNF and k - TDL

We must assume that the examples are randomly sampled from a population on which a fixed but unknown probability measure is defined. Then the two new concept classes defined above can be shown to fit in Valiant's definition of polynomial learnability. To prove this, we will use the theorem by Blumer *et al.* [1] which splits this task into two easier problems. The first is to show that each concept class is polynomial sized - that is the logarithm of its size is a polynomial in the complexity parameters a and f mentioned above. The second is to produce an indentification algorithm which finds a function consistent with the training data or detects its non existence, and is polynomial in a ,f and in the size of the learning set. We get [7] :

- $|k\text{-TDNF}| = 2^{N_k} \leq 2^{k!a^k f^k}$. So $\log(|k\text{-TDNF}|) \in O(N_k) = O((af)^k)$
- $|k\text{-TDL}| = 3^{N_k(N_k!)}$. So $\log(|k\text{-TDL}|) \in O(N_k \log(N_k)) = O((af)^{k+1})$

This shows that both classes are polynomial sized. Furthermore, the identification algorithms are very similar to those used by Valiant [10] and Rivest [9] and have similar complexity (this is detailed in [7]). So these two new classes are polynomially learnable, an important contribution to this result being the polynomiality of the generalisation test.

4 Discussion and conclusion

This paper studies the characteristics of a new example representation language and shows that from a PAC learning point of view, it leads to a complexity similar to that of the Boolean framework. Several points deserve to be discussed :

In our model, the number of possible representations is sometimes infinite (as in our example) which is never the case with Boolean terms. Yet, some aspects of the Boolean framework are preserved, which explains our results concerning PAC learning.

Signatures and grammars have several similarities. Both rest on analogous algebraic construction systems [2]. But our use of signatures is very different to that usually made of grammars. When we place ourselves in a grammatical framework, derivation trees are not essential. Here, on the contrary, we are interested in the trees themselves and give the interior nodes a meaning. Furthermore, the goal of grammatical induction is to learn "the grammar", whereas here we already dispose of the signature. This difference in approach explains our reaching a polynomial learnability result when most problems regarding grammars have a much greater difficulty [5].

We also feel that terms can represent an alternative approach to ILP in the study of recursive rules. Similarities and differences between both approaches remain to be investigated.

References :

1. Blumer, A., Ehrenfeucht, A., Haussler, D. and Warmuth, M.K. (1986). Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*. 273-282.
2. Courcelle, B. (1986). Equivalences and transformations of regular systems - applications to recursive program schemes and grammars, *Theoretical Computer Science* 42 (1), 1-122.
3. Cohen, W.W. (1993). Pac-Learning a Restricted Class of Recursive Logic Programs. *Proceedings of the Tenth National Conference on Artificial Intelligence*. 86-92.
4. Goguen, J.A., Thatcher, J.W., Wagner, E.G. and Wright, J.B. (1977). Initial algebra semantics and continuous algebras, *Journal A.C.M.* 24 (1), 68-95.
5. Gold, M. (1978). Complexity of automaton identification from given data, *Information and control* 37, 302-320.
6. Haussler, D. (1989). Learning conjunctive concepts in structural domains, *Machine Learning* 4,7-40.
7. Jappy, P., Daniel-Vatonne, M.C., Gascuel, O. and de la Higuera., C. (1993). Learning from Recursive, Tree Structured Examples. *Rapport de Recherche LIRMM*. No 93040.
8. Muggleton, S.H. (1992). *Inductive Logic Programming*. Academic Press.
9. Rivest, R.L. (1987). Learning decision list, *Machine Learning* 2(3), 229-246.
10. Valiant, L.G. (1984). A theory of the learnable. *ACM Com. 27*, 1134-1142.
11. Winston, P.H. (1975). Learning Structural descriptions from Examples, in *The psychology of computer vision*, Winston P. H. (Ed.), Mc Graw Hill, New York, 157-209.