ELSEVIER

Artificial Intelligence 117 (2000) 231–253

Artificial Intelligence

www.elsevier.com/locate/artint

Robust logics ☆

Leslie G. Valiant [1]

Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138, USA

Received 14 October 1998

Abstract

Suppose that we wish to learn from examples and counter-examples a criterion for recognizing whether an assembly of wooden blocks constitutes an arch. Suppose also that we have preprogrammed recognizers for various relationships, e.g., on-top-of(x, y), above(x, y), etc. and believe that some possibly complex expression in terms of these base relationships should suffice to approximate the desired notion of an arch. How can we formulate such a relational learning problem so as to exploit the benefits that are demonstrably available in propositional learning, such as attribute-efficient learning by linear separators, and error-resilient learning?

We believe that learning in a general setting that allows for multiple objects and relations in this way is a fundamental key to resolving the following dilemma that arises in the design of intelligent systems: Mathematical logic is an attractive language of description because it has clear semantics and sound proof procedures. However, as a basis for large programmed systems it leads to brittleness because, in practice, consistent usage of the various predicate names throughout a system cannot be guaranteed, except in application areas such as mathematics where the viability of the axiomatic method has been demonstrated independently.

In this paper we develop the following approach to circumventing this dilemma. We suggest that brittleness can be overcome by using a new kind of logic in which each statement is learnable. By allowing the system to learn rules empirically from the environment, relative to any particular programs it may have for recognizing some base predicates, we enable the system to acquire a set of statements approximately consistent with each other and with the world, without the need for a globally knowledgeable and consistent programmer.

We illustrate this approach by describing a simple logic that has a sound and efficient proof procedure for reasoning about instances, and that is rendered robust by having the rules learnable. The complexity and accuracy of both learning and deduction are provably polynomial bounded.

☆ This research was supported in part by grants NSF-CCR-95-04436, NSF-CCR-98-77049, ONR-N00014-96-1-0550, and ARO-DAAL-03-92-G-0115. A preliminary version of this paper appeared in Proc. 31st ACM Symposium on Theory of Computing, Atlanta, GA, May 1999, pp. 642–651.

[1] Email: valiant@deas.harvard.edu.

0004-3702/00/$ – see front matter
PII: S0004-3702(00)00002-3

Keywords: Learning; Reasoning; Deduction; Soundness; Robustness; Binding problem; Learning rules; Learning relations; PAC learning; PAC semantics

1. Introduction

According to Aristotle "every belief comes either through syllogism or from induction" [3]. Computational systems that aspire to exhibit some characteristics of intelligence need to manipulate beliefs about the world. It is reasonable to ask, therefore, how useful Aristotle's dictum is for the construction of such systems. Our purpose here is to argue that the duality expressed by the dictum is fundamental. In particular, we present a formal system that encapsulates this duality and, we believe, offers a vehicle for studying the theoretical basis of such systems.

The history of artificial intelligence can be interpreted as having revolved around this duality from the beginning. Since the 1950s a dominant paradigm, advocated particularly by McCarthy [28], has been that knowledge should be programmed into systems in a uniform logical language as a set of rules, and that logical inference procedures such as syllogisms be used to draw new inferences. As far as learning, Turing had already speculated earlier in 1950 that inductive learning would be used to build machines that think [41]. A few years later he pointed out the computational limitations of logical reasoning alone [42].

In the last forty years much progress has been made both in machine learning as well as in computational logic. Nevertheless, the currently dominant theories of the two phenomena of inductive learning, and of logical reasoning are largely disparate, the notable exception being the area of inductive logic programming [30].

The purpose of this paper is to suggest a formal system that reconciles principled deduction, a characteristic of logic, with robustness, a characteristic of learning. More generally, it encompasses learning and reasoning in a integrated way and retains the somewhat different but crucial benefits that each has to offer.

The particular benefits of mathematical logic that we wish to retain are the existence of a clearly defined semantics for each statement, and the existence of proof procedures that enable new statements to be derived. In particular, the well defined semantics makes possible proof procedures that are *sound*: new statements that are derived from true statements are themselves true.

The main benefits of learning that need to be retained are that it provides a mechanism by which knowledge can be acquired in fragments that may be incomplete, inconsistent or inaccurate, by a system that has no understanding either of its own current global state of knowledge, or of a consistent language of description of the world.

These benefits of learning are somewhat irreconcilable with standard logics, where the rules have to be expressed in terms of a set of predicates with globally consistent meanings. Such enforced global consistency may be achievable in application areas that are known to be amenable to axiomatization, such as much of mathematics. In other areas, such as the world of "commonsense" knowledge, where no such axiomatizations have been found, this approach has led to systems that are "brittle".

The minimal requirements for what we shall call a *robust logic* are the existence of:

(i) Rules to encode knowledge that allow for multiple objects and for relations among them.
(ii) A well defined semantics for the rules.
(iii) An inference procedure for applying rules to instances that is sound and polynomial time.
(iv) Polynomial time algorithms for learning the rules from examples.

Standard predicate calculus suitably constrained, to Horn clauses, for example, satisfies the first three of these requirements. Part (iv) of our definition can be interpreted most simply as claiming that one can make a logic robust if one provides a means of evaluating the accuracy of any statement in it, independently of others, by an empirical process that has access to raw data from the world.

In this paper we describe one particular such robust logic. It arose from efforts to provide theoretical underpinnings for the neuroidal architecture that is described in a companion paper [47]. It has some added benefits, therefore, that derive from that architecture:

(v) The rules can be implemented on a fixed network that mirrors their modularities and dependencies, and the inference procedure can be executed naturally on that network.
(vi) The classes of connective functions that are allowed include a class, namely linear threshold functions, that has ideal learnability properties: there exist efficient learning algorithms for it that are both attribute-efficient and also (according to experimental evidence) error-resilient.

The main departure from traditional logic is that the semantics used here is *PAC semantics* adopted from machine learning. This views inductive learning as a "compu-statistical" phenomenon: observations on the world provide empirical evidence about rules that hold generally in the world. The learner chooses from a space of possible rules according to the weight of statistical evidence, while computational constraints make this choice computationally feasible. This allows robust learning from inaccurate and inconsistent data to be accomplished and to be put on a rigorous foundation. Most basically it is assumed that at any instant the system has some set of primitive sensors, and that its observations of the world can be regarded as induced by a probability distribution D over sets of sensor readings. This distribution D may be arbitrarily complex, reflecting as it does the complexities of the world. In general the system will not need to know the details of D directly. It will, instead, aim to deal with rules that are "simple truths" in the sense that they hold in D, at least with high probability. A system based on this logic may be given rules that are false most of the time, but the system would have the capability of recognizing this fact by making enough empirical observations.

There exist alternative approaches that address aspects of learning and reasoning simultaneously. We shall discuss some of these briefly in Section 4.

The mechanisms that are needed for learning and reasoning in our particular robust logic are computationally efficient. Their complexity depends only polynomially on the number of object variables, on the number of rules in the system, and on their length even when they exploit recursion. The only exponential dependence is on the arity of the relations, which we shall assume to be bounded by a small constant. We believe that this set of requirements, which our system satisfies, is indispensable for any formal system to be viable as a basis for computational intelligence.

2. Scenes

Informally, the objects in our robust logic should be thought of as referring not directly to entities in the world but, instead, to representations in an internal *image* that corresponds to the short-term or working memory of the system. The contents of this image is called a *scene*.

Scenes are defined in terms of a pair $(\mathcal{A}, \widetilde{\Re})$, where $\mathcal{A}$ is a set $\{a_1, \ldots, a_n\}$ of *objects* or *tokens*, and $\widetilde{\Re}$ is a set of *relations* $\{\widetilde{R}_1, \ldots, \widetilde{R}_t\}$ over the objects. The *arity* $\alpha(i)$ of a relation $\widetilde{R}_i \in \widetilde{\Re}$ is the number of its arguments. The set of relations in $\widetilde{\Re}$ that have arity i is called $\widetilde{\Re}^i \subseteq \widetilde{\Re}$. Hence $\widetilde{\Re} = \bigcup \Re^i$ where the union is over a set K of nonnegative integer values of i. For simplicity we shall assume throughout that the sets $\mathcal{A}$, $\widetilde{\Re}$ and all the arities are finite, and define α to be the maximum of the arities $\alpha(i)$ for $1 \leqslant i \leqslant t$.

A *scene* σ is a vector of length $L = L_{\mathcal{A},\widetilde{\Re}} = \sum_{i=1}^{t} n^{\alpha(i)}$. The $n^{\alpha(j)}$ entries after the first $\sum_{i=1}^{i=j-1} n^{\alpha(i)}$ we regard as the jth *group*. These give the truth values of the jth relation $\widetilde{R}_j$ on each of the $n^{\alpha(j)}$ combinations of $\alpha(j)$ objects that can be selected from the n objects in $\mathcal{A}$ as the $\alpha(j)$ arguments of $\widetilde{R}_j$. Thus if $\mathcal{A} = \{a_1, a_2, a_3\}$ and $\widetilde{R}_1$ has arity two, then the vector will have values for all the $3^2 = 9$ entries $\widetilde{R}_1(a_1, a_1)$, $\widetilde{R}_1(a_1, a_2), \ldots, \widetilde{R}_1(a_3, a_3)$, as well as for all the corresponding entries for $\widetilde{R}_2, \widetilde{R}_3, \ldots, \widetilde{R}_t$, etc. Some fixed lexicographic ordering is assumed within each group.

The values in the vector will be from the set $\{0, 1\}$. For the first entry in the above example 1 would mean that $R_1(a_1, a_1)$ is true, and 0 would mean that it is false. The set of such vectors for $(\mathcal{A}, \widetilde{\Re})$ we shall denote by $\Pi_{\mathcal{A},\widetilde{\Re}}$ or Π for short. Clearly $|\Pi| = 2^{\sum_{i=1}^{t} n^{\alpha(i)}}$. We also define the set $\Pi^{\$}$ of *obscured* scenes. In these the vector elements can take a third value \$ that denotes that the corresponding relationship is *obscured*. There are $|\Pi^{\$}| = 3^{\sum_{i=1}^{t} n^{\alpha(i)}}$ such obscured scenes.

We assume that the space Π of scenes for any $(\mathcal{A}, \widetilde{\Re})$ has an associated a probability distribution $D_{\mathcal{A},\widetilde{\Re}}$ over Π. In other words for any scene $\sigma \in \Pi$ the distribution specifies a probability $D(\sigma)$ that a scene drawn randomly from Π will be σ. Clearly such a distribution defines a probability for any condition on scenes. Thus $D(\widetilde{R}_1(a_1, a_2) = 1)$ is the probability that a random scene drawn according to D has $\widetilde{R}_1(a_1, a_2) = 1$. It also attaches a probability to conditions such as $\exists x_1 \forall x_2 \widetilde{R}_1(x_1, x_2)$, namely the sum of $D(\sigma)$ over all σ for which this condition holds.

When considering learnability it is useful to distinguish distributions D over $\Pi_{\mathcal{A},\widetilde{\Re}}$ that are unrestricted from those that are symmetric under permutations of $\mathcal{A}$. For any scene $\sigma \in \Pi_{\mathcal{A},\widetilde{\Re}}$ there exist $n! - 1$ scenes, not necessarily all distinct, that can be obtained from σ by permuting the tokens $a_1, \ldots, a_n \in \mathcal{A}$ in some way. We define distribution D over $\Pi_{\mathcal{A},\widetilde{\Re}}$ to be *symmetric* if any two scenes that can be obtained from each other by permuting the tokens in this way have the same probability. Since our general intention is to regard $\mathcal{A}$ as a set of tokens with no structure we shall assume where not otherwise indicated that all distributions D are symmetric. However, this restriction does not appear to be required for any fundamental purpose and our results apply equally to the asymmetric case, except that the bounds for learnability are larger.

The interpretation of this general framework is as follows. The distribution D is defined, in the spirit of PAC (probably approximately correct) learning, as imposed by

and specifying a possibly arbitrarily complex world. The intelligent system needs to be able to have strategies or concepts that work in this complex world, but the theory of PAC learning suggests that these strategies or concepts can be often learned from examples drawn from D, without the system ever needing to describe D explicitly. The set $\mathcal{A}$ of tokens is intended to correspond to representations in an image internal to the system, as discussed further in [47].

An important general point is that the logic is designed from the start to cope with *partial knowledge*. The relation set $\widetilde{\Re}$ is the space of base relations that the system can recognize at the time in question. For any one scene the truth values of many of these relations will not be explicitly available. In a real world room, for example, every object is made of some material, but in a depiction of the room this information about the materials may not be available, or as we shall say, *specified*. We assume here that D imposes a natural distribution on what knowledge is specified and what is not. Consider the much discussed unary relation that specifies whether an object is a penguin (e.g., [10]). We want that Π distinguish the three cases: true, false and unspecified. For several reasons it is advantageous that the predicates in our logic have just the two values $\{0, 1\}$. In order to allow this in our implementation we expect that the relations are so defined that they *encode* the third value "unspecified" using $\{0, 1\}$. A natural implementation would be to have two unary predicates penguin( ) and penguin*( ) where penguin$^*(a_1) = 0$ would mean that the scene does not specify whether a_1 is a penguin. If the scene does specify whether or not a_1 is a penguin then penguin$^*(a_1) = 1$ would hold and, in that case, penguin$(a_1) = 1$ would denote that a_1 is a penguin and penguin$(a_1) = 0$, that it is not. By having two binary versions of each primitive predicate in this way an element of Π can effectively encode for each primitive predicate and each binding, whether in the scene it is true, false or unspecified. Since scenes are drawn from a probability distribution D over this Π, a probability distribution is imposed over which relation-binding combinations are specified and which are not.

In other words it is acknowledged at the start that in natural situations only partial knowledge is usually available. The fact that learning takes place from situations of partial knowledge can then be offered as an explanation of why the results of PAC learning can be applied effectively to situations of partial knowledge. This provides a principled approach to partial knowledge based on learning as suggested in [45,46]. In practice, it may be that most bindings of most base relations have an unspecified value. In that case, for the sake of succinctness, it is advantageous to represent the scene by listing only those relations and bindings that are specified. This will ensure that the size of descriptions of scenes will be as economical as possible.

We note that the notion of obscured vector elements described earlier is orthogonal to this idea of unspecified vector elements. Once a scene is drawn from D its $\{0, 1\}$ vector elements will determine which relations and bindings are specified and which are not. We can subsequently for the purposes of analysis obscure or hide some arbitrary subset of the vector elements by replacing them with \$ symbols, and so make the scene obscured. For example, when the system performs planning and wishes to evaluate the consequence of a situation, the system will construct scenes where the relations describing the situation are unobscured, but those relating to the consequences are obscured. The deduction process will then evaluate the most likely values of the obscured relations.

In the context of deduction we shall also distinguish between vector elements that are *determined* and those that are not. Initially an obscured element will be undetermined. It becomes determined once a deduction procedure has deduced a value for it.

3. A robust logic

3.1. Syntax

A rule is intended as a statement that in a precise sense that we describe below holds with certainty or some probability for scenes drawn from D. Suppose that scenes are defined for $\mathcal{A} = \{a_1, \ldots, a_n\}$ and $\widetilde{\Re} = \{\widetilde{R}_1, \ldots, \widetilde{R}_t\}$. In describing rules we shall define a set of symbolic relations $\Re = \{R_1, \ldots, R_t\}$ where the arity of R_i is the same as that of $\widetilde{R}_i$. The role of R_i is to model the actual relation $\widetilde{R}_i$. Thus, if in the rule set there is a rule $\forall x\ R_2(x) \equiv R_3(x)$, then the intention is that the corresponding statement $\forall x\ \widetilde{R}_2(x) \equiv \widetilde{R}_3(x)$ should hold with high probability for scenes drawn from D. Also, suppose that in a particular scene σ randomly drawn from D a certain given set of relations from $\widetilde{\Re}$ hold with certain bindings. Suppose further that these relations and bindings, when the corresponding $\Re$ symbols replace the $\widetilde{\Re}$ symbols, are used as premises for the rule set. We want that if it is deduced using the rules that $R_3(a_2, a_4)$, for example, holds, then $\widetilde{R}_3(a_2, a_4)$ should hold with high probability for a randomly drawn scene for which the given set of relations and bindings hold. In other words we want that deductions made using the rules should lead to conclusions that are semantically true for randomly drawn scenes, at least with high probability.

To define the syntax of the rules, besides $\Re$ and $\mathcal{A}$ we also need a set $\mathcal{C}$ of *connectives*, a set X of *object variable* names and some standard symbols. The set of connectives $\mathcal{C} = \bigcup_{i=1}^{\infty} \mathcal{C}^i$ is a set of functions where every $f \in \mathcal{C}^i$ is a Boolean function $f : \{0,1\}^i \to \{0,1\}$.

We shall choose $\mathcal{C}$ so as to have good learning properties. In particular there should be efficient error-resilient and attribute-efficient algorithms for learning its members. Linear threshold functions are a prime example of such a class [4,5,7,26,48]. Attribute-efficiency means that the number of examples grows linearly in the number of relevant variables, but only logarithmically in the irrelevant ones. This is important in the current context in which we envision that variables will be generated automatically in large numbers, for example, to represent relations with all possible bindings, but the number of available examples is limited.

A *rule* q is of the form

$$\forall x_1, \ldots, x_s \in \mathcal{A}\ Q(x_1, \ldots, x_s, \sigma) \to \left[f\left(e_1(R_{i_1}), \ldots, e_\ell(R_{i_\ell})\right) \equiv R_{i_0}(x_1, \ldots, x_s) \right],$$

where $x_1, \ldots, x_s \in X$, $f \in \mathcal{C}^\ell$, and for $0 \leqslant j \leqslant \ell$, $R_{i_j} \in \Re$. Further, for $1 \leqslant j \leqslant \ell$, $e_j(R_{i_j})$ is an *independently quantified expression* (IQE) of the form

$$\triangle_1 y_{j,1}, \ldots, \triangle_{k_j} y_{j,k_j} \in \mathcal{A}\ R_{i_j}(z_{j,1}, \ldots, z_{j,m_j}),$$

where m_j is the arity $\alpha(i_j)$ of R_{i_j}, each $\triangle_h \in \{\exists, \forall\}$ for $1 \leqslant h \leqslant k_j$, each $y_{j,h} \in X$, each

$$z_{j,h} \in \{x_1, \ldots, x_s\} \cup \{y_{j,1}, \ldots, y_{j,k_j}\} \subseteq X$$

and if $y_{j_1,h_1} = y_{j_2,h_2}$ then $j_1 = j_2$. Also Q, the *precondition*, is an arbitrary predicate whose value depends on the scene σ and the binding $\pi : \{x_1, \ldots, x_s\} \to \mathcal{A}$ in question. Its value will depend on which (relation, binding) pairs have values that are unobscured in the input σ, or have been determined by deduction from that input.

Allowing the precondition Q adds to the generality of the framework. For example it may be that we have access to examples of a concept, but only from a certain subdomain that we can characterize by Q. In that case learning a rule with such a precondition is appropriate, especially if there is reason to believe that the concept has no simple definition over a broader domain. Much of our analysis is unchanged, however, if we have the *total* precondition $Q \equiv \mathit{True}$, in which case we shall for brevity suppress that precondition. We impose no general restrictions on Q of learnability or representability. Even if the preconditions are programmed by a human they may have a useful role. Some natural restrictions that would aid mechanical use and warrant further investigation are that Q can be expressed succinctly or computed easily in terms of $\Re$ and $\mathcal{C}$, that Q be symmetric (i.e., invariant under permutations on $\{a_1, \ldots, a_n\}$), and that the truth of Q can be derived within a given set of rules.

A simple example of a rule is

$$\forall x_1, x_2 \in \mathcal{A}\left[\mathit{Th}_2\big(\exists y_1 R_1(x_1, x_2, y_1),\ \exists y_2 R_2(x_1, y_2), \exists y_3 \forall y_4 R_3(x_2, y_3, y_4)\big) \equiv R_4(x_1, x_2)\right],$$

where the total precondition $Q \equiv \mathit{True}$ has been suppressed for brevity. The function Th_2 is the threshold two function of three arguments, defined to be true whenever at least two of its arguments are true. The expression on the left of the $\equiv$ sign is called the *left-hand side* of q, and that on the right the *right-hand* side. These are sometimes abbreviated to $\mathrm{LHS}(q)$ and $\mathrm{RHS}(q)$.

The important constraint in the definition and the example is that the constituent quantified expressions in $\mathrm{LHS}(q)$ are quantified over mutually *disjoint* sets of variable names. In the example these sets are: $\{y_1\}$, $\{y_2\}$ and $\{y_3, y_4\}$. The significance of these constraints is that whenever $f(e_1, \ldots, e_\ell)$ is evaluated on a scene σ for a binding $\pi : \{x_1, \ldots, x_s\} \to \mathcal{A}$, the value of each e_i can be evaluated independently of the others—the values of the y variables that make e_i true have no bearing on the y variables that make e_j true for $j \neq i$. This notion is equivalent to that realized by connection bindings in [47].

This constraint implies, for example, that the relationship expressed by the predicate calculus statement:

$$\mathit{grandfather}(x, y) \equiv \exists z\big(\mathit{father}(x, z) \wedge \mathit{parent}(z, y)\big)$$

cannot be expressed directly as a single rule if $\Re$ contains *father* and *parent*, but not their conjunction. The following pair of rules would, however, suffice:

$$\forall x, y, z \in \mathcal{A}\left[\mathit{father}(x, z) \wedge \mathit{parent}(z, y) \equiv \mathit{grandfather}^*(x, y, z)\right],$$

$$\forall x, y \in \mathcal{A}\left[\exists z\ \mathit{grandfather}^*(x, y, z) \equiv \mathit{grandfather}(x, y)\right].$$

The significance of the disjointness constraint is, therefore, the following. For reasons of computational economy we want to minimize the costs of enumerating bindings. One approach to this is to have relations that use the minimal number of arguments that suffice.

The "grandfather" predicate needs only two if whenever we use it in higher level reasoning, knowledge of the identity of the person z from the intermediate generation is irrelevant. On the other hand, in recognizing the occurrence of the grandfather relationship in the first place we may need to establish the existence of such an intermediate person. Hence at another level we need a relation having this third argument. The above formulation acknowledges both of these realities as economically as possible.

This simple example illustrates that our general approach relies heavily on the assumption that knowledge can be represented with *binding modularity* in terms of relations of small arity. As a second example consider the notion of a dining room. This may be defined in terms of chairs and tables and some relationships amongst them. In turn, a chair may be defined in terms of its constituent parts and the relationships among those. Binding modularity asserts that the arity of the relations needed can be kept small because references to the identities of the parts needed to define the lower level notion, the chair, are redundant when describing the higher level notion of a dining room.

A further issue regarding variable bindings is whether any *inequality constraints* are implied or can be imposed. In particular is it implied that the members of $\{x_1, \ldots, x_s\}$ or $\{y_{j,1}, \ldots, y_{j,k_j}\}$ have to bind to distinct member of $\mathcal{A}$ or not? In our formulation any specific combination of inequality constraints can be made. Thus further notation can be introduced to specify, for example, that x_1 and x_2 must map to distinct tokens, but x_3 need not be distinct from either. The only general prohibition is the one stated earlier, that for *distinct* values j_1, j_2 the bindings of corresponding y_{j_1,h_1} and y_{j_2,h_2} variables can in no way constrain each other.

The following observation will be used throughout.

Fact 3.1. *For any relation R of arity α and any independently quantified expression $e(R)$, the value of $e(R)$ can be computed in $O(n^\alpha)$ steps given the values of R for each of the n^α bindings of its α arguments to $\mathcal{A}$.*

Proof. If in the definition of $e(R)$ we have an order of quantification.

$$\triangle_1 y_1, \triangle_2 y_2, \ldots, \triangle_\alpha y_\alpha$$

then we construct the following rooted game tree of depth α. Each node of the tree at distance i from the root corresponds to a substitution $\{y_1, \ldots, y_i\} \to \mathcal{A}$. Each such node has edges to level $i+1$ nodes corresponding to this substitution, but extended by each of the possible substitutions of y_{i+1} into $\mathcal{A}$. Thus the leaves of the tree are all at depth α and correspond to each of the possible substitutions $\{y_1, \ldots, y_\alpha\} \to \mathcal{A}$.

We now label each leaf by the value of R for the substitution specified by the leaf. We then work up starting from the leaves and attach Boolean values to each node until the root is reached. The value attached to the node corresponding to a fixed substitution $\{y_1, \ldots, y_i\} \to \mathcal{A}$ will be the truth value of

$$\triangle_{i+1} y_{i+1}, \ldots, \triangle_\alpha y_\alpha R(),$$

where $R()$ denotes the relation R when the given fixed substitution has been made. It is easy to verify by induction that to label a node at distance $i-1$ from the root where $\triangle_i = \exists$,

one takes the disjunction of the labels of its son nodes, and where $\triangle_i = \forall$ one takes the conjunction of the labels of its son nodes.

Clearly, a similar procedure works if some inequality constraints are imposed such as $y_2 \neq y_1$.

Also, the procedure works if in the definition of $e(R)$ some arguments have fixed values in $\mathcal{A}$, and quantification occurs only on the subset of remaining variables. □

We note that existential quantifications allow the above algorithm to successfully label the root node even in some cases when the relation R is not determined for some of the bindings (e.g., it corresponds to an obscured value in the input, and no deduction has yet been performed) and the corresponding leaves in the game tree have no labels. We shall say that $e(R)$ is *determined* whenever the procedure succeeds in giving the root node a valid label. We note also that when we discuss deduction in Section 3.4 we shall entertain the possibility that a fixed R with a fixed binding evaluates to both of the values 0 and 1, in which case the value is written as $0/1$. The game tree evaluation procedure will be adapted so that whenever this is the case, it will evaluate the consequences of both a 0 value and also of a 1 value.

To summarize, therefore, a leaf can have one of four values: 0,1, undetermined or $0/1$. The labelling procedure can attach any of these four values to internal tree nodes also as it proceeds.

When evaluating an expression e by means of the procedure of Fact 3.1 we shall therefore output

(i) undetermined if no value combination of the leaf values determines a 0 or 1 value for e,
(ii) a 0 value if some combination determines 0, and none determines a 1,
(iii) a 1 if some combination determines a 1 and none a 0, and
(iv) $0/1$ otherwise.

An important note is that this process is *monotone* in the sense that once e is determined, subsequently changing an undetermined input value to determined, or a determined one to $0/1$ will never make e undetermined.

The *size of description* $\|q\|$ of a rule q will be the number of occurrences of relation symbols in its definition. The *size of description* $\|S\|$ of a set S of rules will be the sum of $\|q\|$ over all $q \in S$. The *size* $|S|$ of S will be the number of rules in S.

A rule set S is *acyclic* if its graph $G(S)$ is acyclic, where $G(S)$ is defined in the following manner: The graph $G(S)$ is a directed graph $G = (V, E)$ where V, the set of nodes, is the set $\Re$, and E, the set of edges is a subset of $\Re \times \Re$. In particular, the directed edge $(R_i, R_j) \in E$ if and only if there is a rule in S that contains R_i on the left-hand side and R_j on the right-hand side. We denote the class of acyclic rule sets by Γ, and the class of arbitrary rule sets by Γ^*. For $\alpha = 1, 2, \ldots$ we denote by Γ_α and Γ_α^* these classes restricted to rule sets where the maximum arity of any relation is α.

We allow more than one rule to share the same $R \in \Re$ on the right hand side. This allows for multiple definitions of R, possibly with different preconditions and depending on different relation sets on the left hand side. Note that if two rules have the same R on the right hand side and have some R_i in common on the left hand side then some edge in

$G(S)$ will have arisen from more than one rule. Of course, multiple rules may give rise to mutually contradictory deductions.

It will turn out that our results apply equally to general graphs as to acyclic ones. The importance of allowing general graphs is that recursive rules can then be expressed. For example the earlier expressions given can be adapted to:

$$\forall x, y, z \in \mathcal{A}\left[\text{ancestor}(x, z) \wedge \text{parent}(z, y) \equiv \text{ancestor}^*(x, y, z)\right],$$

$$\forall x, y \in \mathcal{A}\left[\exists z\ \text{ancestor}^*(x, y, z) \equiv \text{ancestor}(x, y)\right].$$

The rules we have defined can be compared with Horn clauses in first order predicate calculus. The implication sign has been replaced by equality, and conjunctions on the left hand side have been generalized to wider classes of connectives, including linear thresholds, that can make learning more tractable and robust. On the other hand, for the sake of controlling computational complexity some restrictions are imposed also. A restriction that is fundamental and shared, for example, with the datalog model for databases [43], is that the function symbols of predicate calculus are excluded in order to prevent the proliferation of the objects that are quantified over. This restriction enables the complexity of the binding problem to be controlled. For example, if we had a constant "Napoleon" and a function "parent" then expressions of the form "parent^i(Napoleon)" would refer to an unbounded number of individuals, namely the ancestors of Napoleon. Another difference is that instead of having constants to refer to individuals, we instead use unary predicates. Thus instead of the constant "Napoleon", we have Napoleon(a_i), which corresponds to the token a_i qualified by a unary relation, representing that individual.

3.2. Semantics

In the previous section we defined the syntax of rules and rule sets in terms of a triple $\Re$, $\mathcal{A}$ and $\mathcal{C}$ where $\Re$ is a set of relations, $\mathcal{A}$ a set of tokens, and $\mathcal{C}$ a set of connective functions. Now, for any scene $\sigma \in \Pi_{\mathcal{A},\widetilde{\Re}}$ and any binding $\pi : \{x_1, \ldots, x_s\} \to \mathcal{A}$, the rule q

$$\forall x_1, \ldots, x_s\ Q(x_1, \ldots, x_s, \sigma) \to \left[f\left(e_1(R_{i_1}), \ldots, e_\ell(R_{i_\ell})\right) \equiv R_{i_0}(x_1, \ldots, x_s)\right]$$

can be examined for the truth values of each $e_j(R_{i_j})$ and hence also of both the left-hand side

$$f\left(e_1(R_{i_1}), \ldots, e_\ell(R_{i_\ell})\right),$$

and the right-hand side

$$R_{i_0}(x_1, \ldots, x_s).$$

We define the $\{0, 1\}$-valued functions $lhs(q, \sigma, \pi)$ and $rhs(q, \sigma, \pi)$ as follows: The former function $lhs(q, \sigma, \pi) = 1$ if and only if binding π makes f true on σ after the $\widetilde{\Re}$ relations are substituted for the $\Re$ relations. The latter function $rhs(q, \sigma, \pi) = 1$ if and only if $\widetilde{R}_{i_0} = 1$ on σ with binding π.

We can then define the *false positive* and *false negative* $\{0, 1\}$-valued error functions $err^-(q, \sigma, \pi)$ and $err^+(q, \sigma, \pi)$ as follows: $err^+(q, \sigma, \pi) = 1$ if and only if: $Q(\pi, \sigma) = 1$, $lhs(q, \sigma, \pi) = 1$ and $rhs(q, \sigma, \pi) = 0$, and similarly $err^-(q, \sigma, \pi) = 1$ if and only if:

$Q(\pi,\sigma)=1$, $lhs(q,\sigma,\pi)=0$ and $rhs(q,\sigma,\pi)=1$. Note that since Q depends on the scene σ and the binding $\pi=\{x_1,\ldots,x_s\}\to\mathcal{A}$, we can with only slight abuse of notation express $Q(x_1,\ldots,x_s,\sigma)$ as $Q(\pi,\sigma)$.

Finally, we can define the probability that the rule q will predict false positives or false negatives on scenes σ drawn randomly from $\Pi=\Pi_{\widetilde{\Re},\mathcal{A}}$ according to distribution D, as follows:

$$er_D(q)=\max_{\pi}\sum_{\sigma}D(\sigma)\big(err^+(q,\sigma,\pi)+err^-(q,\sigma,\pi)\big).$$

We note that we are giving each scene σ its arbitrary probability as defined by D, but restrict the sum to those σ that satisfy Q. Also, since we require small error for *every* binding, we define the error measure as the maximum of the errors over all the $s=n^{\alpha(i_0)}$ bindings. Note that we can define analogously er_D^+, er_D^- as er_D above but omitting the err^- and err^+ term respectively. As a summarizing simple measure of accuracy, however, we use er_D and note that in the case that D is symmetric the maximization over π in the definition can be replaced by the substitution of any fixed π.

Definition. A rule q is ε-*accurate* in D if and only if $er_D(q)\leqslant\varepsilon$.

3.3. Learnability

The goal of our logic is to provide a framework in which we can reason from knowledge that is learned. Hence central to it is some minimal notion of learnability for rule sets. This minimal notion will capture the idea that if we have a complete specification of a rule *except* for the identity of the connective $f\in\mathcal{C}$, then an approximation to f can be learned in the PAC sense from random examples of scenes $\sigma\in\Pi$ drawn according to distribution D. Clearly a complete specification requires specifications of the identities of and the quantifiers for relations $R_{i_0},R_{i_1},\ldots,R_{i_\ell}$. It also requires specifications for each of the $\alpha(i_h)$ argument positions in R_{i_h}, if any, of the arguments $x_1,\ldots,x_s$ of R_{i_0} that are to be substituted there. The remaining arguments are all distinct for the various values of h $(1\leqslant h\leqslant l)$. There may be additional inequality constraints in the specification.

Definition. For a set of rules where each rule q is of the form

$$\forall x_1\ldots,x_s\in\mathcal{A}\ Q(x_1,\ldots,x_s,\sigma)\\ \to\big[f\big(e_1(R_{i_1}),\ldots,e_\ell(R_{i_\ell})\big)\equiv R_{i_0}(x_1,\ldots,x_s)\big] \tag{3.1}$$

over $\Re,\mathcal{C},\mathcal{A}$, consider learning algorithms that in unit time can obtain for any desired binding π a scene randomly chosen from the distribution D restricted to scenes that satisfy $Q(\pi,\sigma)$. We say that a class of rules is *PAC-learnable from scenes* if there is a learning algorithm that for any rule q of the form (3.1), for any $\varepsilon>0,\delta>0$ and any symmetric D, runs in time a fixed polynomial in $1/\varepsilon,1/\delta,n$ and the size of description of q, and outputs with probability at least $(1-\delta)$ an $f'\in\mathcal{C}$ that makes the rule ε-accurate when f' is instantiated in place of f. [The oracle for $Q(\pi,\sigma)$ may be called with an arbitrary requested binding π and will return an unobscured σ randomly chosen according to D

from those satisfying $Q(\pi,\sigma)$. If $Q(\pi,\sigma)$ is false for all σ or for all σ having non-zero probability in D, then an arbitrary f' may be returned.]

In other words we will be seeking to learn a rule of a fixed form, such as the following:

$$\forall x_1,x_2,x_3\ Q(x_1,x_2,x_3,\sigma) \rightarrow \big[f(\exists y_1 R_1(x_1,x_2,y_1),\exists y_2 R_2(x_2,x_3,y_2)) \equiv R_1(x_1,x_2,x_3)\big].$$

The algorithm will seek to find an f' that when substituted in place of f in this fixed form will hold with high probability for random examples satisfying Q. Note that the restriction of f to a fixed form ceases to be a true restriction if the enumeration $e_l,\ldots,c_\ell$ is a complete enumeration of all the base relations with all bindings, as we envisage to be viable if $\mathcal{C}$ is learnable attribute-efficiently.

In the symmetric case we have the following.

Theorem 1. *If $\mathcal{C}$ is PAC-learnable as a class of Boolean functions by an algorithm M for which a number $L(\ell,\varepsilon,\delta)$ of examples is sufficient to learn $\mathcal{C}^\ell$ to accuracy ε with confidence $1-\delta$, then for any $\Re$ and $\mathcal{A}$, any class of rules with constant arity over $\mathcal{C}$, $\Re$ and $\mathcal{A}$ is PAC-learnable from scenes. Also $L(\ell,\varepsilon,\delta)$ examples suffice to learn to accuracy ε with confidence $1-\delta$ if D is symmetric.*

Proof. We assume that we have a rule

$$\forall x_1,\ldots,x_s\ Q(x_1,\ldots,x_s,\sigma) \rightarrow \big[f\big(e_1(R_{i_1}),\ldots,e_\ell(R_{i_\ell})\big) \equiv R_{i_0}(x_1,\ldots,x_s)\big] \quad (3.2)$$

and that full descriptions of $e_h(R_{i_h})$ for $1\leqslant h\leqslant l$ are known, as is also the identity of R_{i_0}. Given a set of scenes $\sigma\in\Pi_{\mathcal{A},\widetilde{\Re}}$ drawn according to D, the task of the learning algorithm is to derive an $f'\in\mathcal{C}^\ell$ that when substituted for f in (3.2) will give a rule that with confidence $1-\delta$ is ε-accurate. The constant arity bound ensures that, by Fact 3.1, the value of each $e_h(R_{i_h})$ can be evaluated in polynomial time for any scene.

Since D is symmetric it will suffice to learn with one fixed binding, e.g., $\pi^*(x_i)=a_i$ for $1\leqslant i\leqslant s$. We shall assume, as is sufficient for the statement of the Theorem, that the algorithm has access to a source of examples σ all of which satisfy $Q(\pi^*,\sigma)=1$ for the chosen binding π^*. For each input scene σ we shall consider the truth value of $\widetilde{R}_{i_0}$, and of each $e_j(\widetilde{R}_{i_j})$ for $1\leqslant j\leqslant\ell$, when the substitution π^* has been made in these expressions. This will give us a vector of l truth values as an input for f, and a Boolean value as output.

By assumption the learning algorithm M for $\mathcal{C}$ can learn an $f'\in\mathcal{C}$ that is ε-accurate with confidence $1-\delta$ from $L(\ell,\varepsilon,\delta)$ examples in time polynomial in $\varepsilon^{-1},\delta^{-1},\ell$, and the size of description f. But this is equivalent to saying that with confidence $1-\delta$ the rule q' that is learned (i.e., rule q with f' substituted for f) has the property that:

$$er_D(q',\pi^*)=\sum_\sigma D(\sigma)\big(err^+(q',\sigma,\pi^*)+err^-(q',\sigma,\pi^*)\big)\leqslant\varepsilon.$$

Since D is symmetric the quantity $er_D(q',\pi^*)$ is invariant under the choice of π^*. Since $e_D(q)$ is defined in (3.3) to be the maximum of $er_D(q,\pi)$ over all π, we conclude that $er_D(q')\leqslant\varepsilon$. □

If D is not symmetric learnability can still be defined and the learning problem is not fundamentally more difficult. However, in general, one would then have to learn a distinct f'_π for each distinct

$$\pi : \{x_1, \ldots, x_s\} \to \mathcal{A}.$$

The oracles for $Q(\pi, \sigma)$ defined in the definition of PAC-learnability of rules provides for the source of examples needed for each such π. However, the sample complexity would potentially increase by a factor of n^s.

The following is a brief discussion of the intentions behind the preconditions Q: Clearly preconditions have a number of valuable uses. For example, it may be that a simple rule for, say, $R(x)$ exists or is known only for a certain subdomain Q. Alternatively, it may be that a small number of subdomains cover most of the positive occurrences of R, and distinguishing these contexts explicitly is the best way of encoding the relevant information. How constraining should the preconditions be allowed to be? If we regard the distribution D as referring to a base domain Q_0, then our definitions are intended, in the first instance, to refer to the case that the Q for each rule covers a significant part (i.e., at least inverse polynomial) of D. If that holds then in our definition of PAC-learnability of rules the assumption that the examples oracle has to produce examples that satisfy Q can be eliminated, since we could instead simply discard examples that did not satisfy Q. For any rule set it will be most convenient to define Q_0 to be the union of the preconditions of all the rules in the set, assuming that each rule does have a satisfactory precondition. Philosophically, one can hypothesize a universal precondition Q^* and a distribution D^* over that into which every Q_0 and D can be embedded. In practice it appears to be preferable to discuss the minimal Q_0 and D that apply to a given rule set, rather than the universal Q^* and D^*.

As mentioned in an earlier section, it is an important fact that for $\mathcal{C}$ one can choose useful classes that are learnable attribute-efficiently: Since the number ℓ of arguments of f will grow as n^α if all possible quantified expressions are used as arguments it will be desirable that the sample complexity be smaller than this quantity. This is possible, for example, if $\mathcal{C}$ is chosen to be Boolean disjunctions with few terms or Boolean functions defined by linear inequalities with small integer coefficients and few terms [26,48].

We note that the exact number of possible quantified expressions can be written as follows, if inequality constraints are not allowed: If R_{i_0} on the right-hand side has arity s then for any relation of arity $\beta \leqslant \alpha$ the number of expressions in which i arguments are quantified is $2^i s^{\beta-i} \binom{\beta}{i}$. Summing this for $i = 0, \ldots, \beta$ gives $(s+2)^\beta$.

3.4. Deduction: Soundness and completeness

In this section we discuss algorithms for making deductions about obscured scenes by means of a set S of rules that are to be thought of as capturing some relevant knowledge. We restrict ourselves to deduction algorithms that are efficient in the sense that they require only polynomial time. The other crucial property that is desired is that the deduction algorithm be sound, in the sense that any conclusion reached should be true, at least with high probability, if the constituent rules were accurate.

We shall use the following definitions:

Definition. A *deduction algorithm* takes as input
(i) a set S of rules with respect to some $\Re$, $\mathcal{A}$, and $\mathcal{C}$,
(ii) a relation $R_i \in \Re$ that is the right-hand side of some $q \in S$,
(iii) a binding π from the arguments of R_i to $\mathcal{A}$, and
(iv) a scene $\sigma \in \Pi^{\$}_{\Re\mathcal{A}}$ obscured by the replacement by \$ of various values including that of R_i with binding π.
The deduction algorithm outputs either a predicted $\{0, 1, 0/1\}$ value for R_i on σ with binding π, or a special "nothing predicted" symbol.

Note that this definition allows a deduction algorithm to output "nothing predicted" all the time. In Section 3.4.3 we shall introduce the notion of completeness that will be used to disallow misuse of this possibility.

Definition. A deduction algorithm is *polynomial time* if it runs in time bounded by a fixed polynomial in: the *size of description* $\|S\|$ of the rule set S, the size of description of the scene σ, the size $n = |\mathcal{A}|$ of $\mathcal{A}$, and the maximum computational complexity $c(S)$ of the connectives from $\mathcal{C}$ appearing in S.

Definition. A deduction algorithm is ε-*accurate* on a rule set S and distribution D if for any $R_i \in \Re$ and π, any predicted $\{0, 1, 0/1\}$ value of R_i on binding π is incorrect with probability at most ε on a scene σ drawn randomly according to D.

In order to interpret this definition note that for any R_i and π, if σ is drawn from D then one of three things can happen:
(a) there is a predicted value of R_i and π that is incorrect (i.e., a value of $0/1$ is always taken as incorrect, while a 0 or 1 value is incorrect if it differs from $\widetilde{R}_i$ on π and σ),
(b) there is a predicted 0 or 1 value that is correct, and
(c) there is no predicted value.
The above definition therefore insists that eventuality (a) occurs with probability at most ε.

It is instructive to subdivide (c) further into two subcases. In subcase (c1) some prediction would have been made if all the rules had the total precondition, but some of the actual preconditions were too restrictive to apply. In the second subcase (c2) no prediction would have been made even if all the preconditions had been total. This latter subcase can arise for example, if some needed values are obscured in σ, or if the rules are such that they never have implications for R_i.

Definition. A deduction algorithm is *PAC-sound* if (i) it is polynomial time and (ii) for some polynomial $p(\,)$, for every D, for every $\varepsilon > 0$, it is ε-accurate for any rule set S in which each rule is $p(\varepsilon/(\|S\||\mathcal{A}|))$-accurate, where $|\mathcal{A}|$ is the number of elements of $\mathcal{A}$, and $\|S\|$ is the size of description of S.

Note that while ε-accuracy may be a weak requirement on rule sets S that rarely produce predictions, PAC-soundness when applied to procedures that are complete is a strong one.

3.4.1. The acyclic case

We now define the *acyclic deduction algorithm* which is to be applied to any acyclic rule set S with respect to any $\Re, \mathcal{A}$ and $\mathcal{C}$ on any scene $\sigma \in \Pi^{\$}_{\widetilde{\Re},\mathcal{A}}$.

The algorithm first constructs the graph $G(S)$ of the rule set, and then renumbers the relations R_i, and associated $\widetilde{R}_i$, so that they are topologically sorted, i.e., for every directed edge (R_i, R_j) in $G(S)$ it is the case that $j > i$. At each node R_j there is maintained a table that contains entries for each of the $n^{\alpha(j)}$ possible bindings of R_j to $\mathcal{A}$. Each entry has a value from $\{0, 1, 0/1, ?\}$. Initially all the values that are unobscured in σ are entered as the appropriate 0 or 1 values in these tables. All the remaining entries are made "?".

The algorithm considers in turn the $|S|$ rules in order of increasing j where R_j is their right hand side, the relative ordering of rules with the same R_j on the right-hand side being arbitrary:

$$\forall x_1, \ldots, x_s \in \mathcal{A}\ Q(x_1, \ldots, x_s, \sigma) \to \left[f\big(e_1(R_{i_1}), \ldots, e_\ell(R_{i_{\ell(j)}})\big) \equiv R_j \right].$$

When considering the current rule with RHS $= R_j$, the algorithm is in a position to do the following for *each* of the n^s bindings π of $x_1, \ldots, x_s$ to $\mathcal{A}$ where $s = \alpha(j)$:

It tests whether $Q(x_1, \ldots, x_s, \sigma) = 0$ and if that is so it does no update. Otherwise for each R_{i_h} $(1 \leqslant h \leqslant \ell(j))$, and for every binding π' of its y variables (i.e., those arguments of R_{i_h} not bound by π) it takes the value of R_{i_h} (which may be determined either by being unobscured in the input, or by having been evaluated) under the combined binding π, π', and hence evaluates $e_h(R_{i_h})$ using Fact 3.1. If the values of R_{i_h} needed to evaluate some $e_h(R_{i_h})$ include "?" values in such a way that e_h cannot be evaluated for this π then the entry for π at R_j will not be updated. Otherwise substituting these $\ell(j)$ values of the e_h into f gives the deduced value of R_j under the binding $\pi : x_1, \ldots, x_s \to \mathcal{A}$, and will be entered as the value under π in the table at node R_j. If R_j occurs on the right-hand side of more than one rule then for any one π it may acquire a 0 value through one rule, and a 1 value through another. In that case the table entry made for it will be $0/1$. Now if some R_{i_h} in the rule under current consideration occurs on the RHS of more than one rule and it has acquired a $0/1$ value thereby for some π, then e_h may also acquire such a $0/1$ value through the execution of the procedure of Fact 3.1. The algorithm will then choose just *one* $\{0, 1\}^\ell$ vector consistent with the values of the e_h, say the lexicographically first one, and evaluates the connective f for just that one vector.

When the deduction algorithm terminates it outputs the table entry for the (relation, binding) pair for which a value was requested, and if that value is ? then it outputs "nothing predicted".

Theorem 2. *For any fixed α the acyclic deduction algorithm is PAC-sound for acyclic rule sets Γ_α composed of relations of maximum arity α.*

Proof. The algorithm repeats the action described in the last paragraph $n^{\alpha(q)}$ times for each rule $q \in S$. Each such action evaluates $\ell(q)$ values of $e_h(R_{i_h})$ for the various h and performs one evaluation of a connective function. Hence the overall complexity of the algorithm subsequent to the initialization of the tables is upper bounded by

$$\sum_q n^{\alpha(q)} \big(\mathrm{Ev}(n, \alpha) l(q) + c(q)\big),$$

where $c(q)$ is the complexity of evaluating the connective f in rule q and $\mathrm{Ev}(n,\alpha)$ is the complexity of evaluating a quantified expression for a relation of arity α over an image with n object variables. If we let $\|S\|$ be the length of description of S as defined earlier then $\|S\| = \sum_q (\ell(q)+1)$. Hence if we denote by $c(S)$ the maximum complexity of the connectives in S and use Fact 3.1 to upper bound $\mathrm{Ev}(n,\alpha)$ by $\mathrm{O}(n^\alpha)$, then the runtime of the algorithm can be upper bounded by a term linear in the description of the input σ, plus a term linear in

$$n^\alpha\big(\|S\|n^\alpha + |S|c(S)\big).$$

This establishes that the algorithm is polynomial time if α is regarded as a constant.

We now show that the algorithm is sound in the sense that if each rule of S is accurate enough over D then any output from the algorithm will be inaccurate with small probability on a random scene σ drawn according to D.

Consider an acyclic rule set S and a scene σ for it. As the algorithm evaluates S it evaluates at most $|S|$ rules and considers them in sequence. When evaluating the connective f for the rule q for some binding π for each constituent R_{i_h} on the left-hand side it uses the values of R_{i_h} for the various different bindings consistent with π that had been evaluated when the rules with R_{i_h} on the right-hand side had been computed.

Now consider the event that σ drawn randomly according to D has some $\widetilde{R}_j = 0$ for some binding π, but that the deduction algorithm outputs $R_j = 1$ for that binding, or, alternatively, $\widetilde{R}_j = 1$ but the algorithm outputs $R_j = 0$. Then it must be that there is a first rule and binding in the topological ordering that make a mistake, i.e., the deduced values of every R_{i_h} on the left-hand side do equal the real values $\widetilde{R}_{i_h}$ but the connective f produces an answer different from the true value of the right-hand side. But the quantity

$$er_D(q) = \max_\pi \sum_\sigma D(\sigma)\big(err^+(q,\sigma,\pi) + err^-(q,\sigma,\pi)\big)$$

upper bounds the probability that on any π a random σ from D satisfies Q and causes rule q to err. Let us assume, as is sufficient for the definition of PAC-soundness, that this probability is at most $\varepsilon/(|S|n^\alpha)$. Then the probability that a random σ will cause at least one rule in S to err on at least one binding is no more than $|S|n^\alpha$ times this quantity, namely ε. This establishes that the deduction algorithm is PAC-sound. □

3.4.2. The general case

We now describe the *general deduction algorithm* that can be applied to arbitrary rule sets, not just acyclic ones.

For an arbitrary rule set S we define a directed *evaluation graph* $\mathrm{EG}(S) = (V, E)$ as follows. We assume that the rules are defined in the manner of (3.1) and ordered arbitrarily $j = 1, \ldots, |S|$. We denote the right hand side of the jth rule by $R_{r(j)}$. More than one rule may share the same RHS in which case r is not one-to-one. The vertex set V of $\mathrm{EG}(S)$ is:

$$V = \bigcup_{j=1}^{|S|} \big\{(r(j),\pi) \mid \pi : \{x_1, \ldots, x_{\alpha(j)}\} \to \mathcal{A}\big\}.$$

In other words, each vertex corresponds to a relation $R_{r(j)}$ and to a binding of its arguments to $\mathcal{A}$. Hence certainly $|V| \leqslant |S| \cdot n^{\alpha(S)}$. With regard to the edges E of $\mathrm{EG}(S)$

each rule j is associated with a (not necessarily disjoint) set of directed edges (v_g, v_h) where $v_h = (r(j), \pi)$ for some π, and $v_g = (k, \pi')$ where, further, R_k appears in LHS(j). More precisely, the presence of (v_g, v_h) in E denotes that in order to evaluate $R_{r(j)}$ for π using LHS(j) the value of R_k for π' *may* be needed. For example, suppose that rule $j = 5$ is:

$$\forall x_1 \in \mathcal{A}\left[f\left(\exists y_1\, R_1(x_1, y_1), \exists y_2\, R_2(y_2, x_1)\right) \equiv R_3(x_1)\right]. \tag{3.3}$$

Also let us denote the binding $\pi : \{x_1, \ldots, x_s\} \to \{a_{\pi(1)}, \ldots, a_{\pi(s)}\}$ by $[\pi(1), \ldots, \pi(s)]$. Then if $v_h = (3, [4])$ and $v_g = (2, [5, 4])$ then the edge (v_g, v_h) will be present in E since to verify that $R_3(a_4) = 1$ we may need the value of $R_2(y_2, a_4)$ for *every* value of $y_2 \in \mathcal{A}$, including $y_2 = a_5$.

Whether the value of this $R_2(y_2, a_4)$ is needed in a particular evaluation depends on the details of the actual implementation. For example, suppose that the value of $R_2(a_7, a_4)$ has been previously obtained in the course of the evaluation. If its value was 1 then this determined the value of $\exists y_2 R_2(y_2, a_4)$ already, and hence the value of $R_2(a_5, a_4)$ would no longer be needed. If its value was 0, on the other hand, then the value of $R_2(a_5, a_4)$ may still be necessary to determine the value of $\exists y_2 R_2(y_2, a_4)$.

The in-degree of each node is determined very simply from the syntax. In the example described in expression (3.3) the node $(3, [i])$ has $2n$ predecessors deriving from this rule, a half from nodes $(1, [i, j])$ for $1 \leqslant j \leqslant n$, and the other half from $(2, [k, i])$ for $1 \leqslant k \leqslant n$. If R_3 appears as the RHS of other rules then the in-degree would be larger. More formally, the predecessors in EG(S) of a node (j, π) are the (i, π') with the following property: for some rule with R_j on the right-hand side there occurs R_i on the left-hand side, and for the x variables that occur as common parameters of R_j and R_i in the rule, the bindings π and π' map them to the same $a \in \mathcal{A}$.

We shall regard a deduction algorithm as a procedure that dynamically assigns a label from $\{0, 1, 0/1, ?\}$ to each node of EG(S). A node that has one of the first three labels 0,1, $0/1$ is considered *set*, while a node labelled ? is considered *unset*. Initially each node that corresponds to a value that is *not* obscured in the scene σ is set to the $\{0, 1\}$ value specified by σ. All other nodes are initially given the unset value ?. The label value of a node v is changed from ? to 0 or 1 only if the labels of the predecessors of v in EG(S) are such that they fully determine enough arguments of f so as to determine f. In the example described in (3.3), for $\pi : x_1 \to a_4$ both $\exists y_1 R_1(a_4, y_1)$ and $\exists y_2 R_2(y_2, a_4)$ need to be determined if, for example, f is the parity function. Note that $\exists y_1 R_1(a_4, y_1)$ is determined if either $R_1(a_4, a_i)$ is set to 1 for *some* $a_i \in \mathcal{A}$, or $R_1(a_4, a_i)$ is set to 0 for all $a_i \in \mathcal{A}$. When a node in EG(S) is set we will, for the purposes of the algorithm description, consider that the corresponding vector element in σ that was originally obscured now has that set value.

Clearly, once the label of a node is set to 0 or 1 by virtue of a rule, its label will not be further changed even if more of the predecessor nodes arising from the same rule are set to $\{0, 1\}$ values subsequently. However, if the node corresponds to an R_i that occurs on the RHS in more than one rule then a contradiction that tries to set a value of 0 to one already set to 1, or vice versa, may arise. In that case the algorithm will set the value "$0/1$". As in the acyclic case, for each node we shall compute the connective f just the first time that its value is defined, and if some argument then has a $0/1$ label we will choose one of these arbitrarily.

The *general deduction algorithm* we can now define as follows: The rules are ordered in an arbitrary manner $j = 1, \ldots, |S|$. The labels of the nodes of $\mathrm{EG}(S)$ that have unobscured 0 or 1 values for the given σ are set to those values. The algorithm then *scans* the rule set repeatedly and terminates when a scan of S has been completed in which no *updates* occurred. In each scan it enumerates the rules $j = 1, \ldots, |S|$ and considers each one for updating. In each such consideration it enumerates all the bindings $\pi : \{x_1, \ldots, x_{\alpha(r(j))}\} \to \mathcal{A}$: For each such binding, if the precondition Q of that rule is satisfied by (π, σ), and if the label for the node $(r(j), \pi)$ in $\mathrm{EG}(S)$ should be changed because the rule is now fully determined by the labels of its predecessors in $\mathrm{EG}(S)$ while this was not the case in the previous scan, then the algorithm evaluates f and performs the necessary update to the label. (In the case that a 0 or 1 value would be assigned to a node that has been set a contrary 0 or 1 value previously by virtue of a different rule, the set value will be changed to "0/1".)

When the algorithm terminates it considers the value of the label of the relation R_i for the binding π specified as the deduction task, and outputs that value. If it is ? then it outputs "nothing predicted".

Theorem 3. *For any fixed α the general deduction algorithm is PAC-sound for rule sets Γ^*_α composed of rules of arity upper bounded by α.*

Proof. Each (relation, binding) pair (i, π) can partake in at most two updates—when it is first set, and when it is changed to 0/1. Since at least one (i, π) pair has an initialized label at the start, and at least one is updated at each step it follows that there are at most $2tn^\alpha$ scans if there are t relations and therefore t choices of i, and there are n^α choices of π.

Each scan considers each node (i, π) of $\mathrm{EG}(S)$ as many times as there are rules j with $r(j) = i$. Hence it performs at most $|S|n^\alpha$ such considerations. In each consideration it examines each of the ℓ_j expressions in the LHS and evaluates it, if necessary, using $\mathrm{Ev}(n, \alpha)$ operations, and if all these yield values then it computes the connective f.

Hence a first upper bound on the runtime is

$$2tn^\alpha\big(n^\alpha \|S\| \mathrm{Ev}(n, \alpha) + n^\alpha |S| c(S)\big).$$

Now it is the case, if one considers the algorithm globally, that the number of times connectives need to be evaluated is just the number of (i, π) pairs, or at most $|S|n^\alpha$. Also, if quantified expressions are evaluated using a game-tree in the manner of Fact 3.1, then a reduced bound on the total work needed can be deduced using global considerations. Suppose we modify the algorithm so that whenever a node (i, π) is updated the internal nodes in all the game trees in which (i, π) occurs as a leaf are also modified, and their effects followed up towards the root of the game tree. Following the effects of any one leaf costs at most α operations, and therefore the accumulated updating done on any one game tree is at most αn^a operations.

Using these two global considerations one can reduce this complexity bound to $2\alpha t n^{2\alpha} + |S|n^\alpha c(S)$.

The additional cost of initializing the labels is linear in this quantity also since it is linear in the number of nodes $|S|n^\alpha$ of $\mathrm{EG}(S)$. Hence the algorithm satisfies the polynomial time criterion if α is a constant.

For analyzing soundness suppose that each rule is ε'-accurate. This means that for any one binding π of its RHS, for a σ randomly drawn from D it is the case that $Q(\sigma,\pi)=1$ and LHS $\neq$ RHS with probability at most ε'.

Now in the course of an evaluation the general deduction algorithm performs at most $|S|n^{\alpha}$ label updates. If the output of the algorithm is incorrect for the given σ then it must be that in one of the label updates, say for rule j under binding π', the arguments of $\mathrm{LHS}(j)$ as computed were correct for σ, but applying the function f specified in rule j produced a false value for $\mathrm{RHS}(j)$. However, for any such fixed j and π' the probability of this occurring is at most ε' since the rule is ε'-accurate by assumption.

Hence the probability that any false conclusions are drawn by the algorithm is at most $|S|n^{\alpha}\varepsilon'$. If $\varepsilon' < \varepsilon/(|S|n^{\alpha})$ then it follows that the output of the general deduction algorithm will be incorrect with probability at most ε, as required. □

3.4.3. Completeness

It seems clear that in the definition of ε-accuracy we need to allow for the possibility that the deduction algorithm may legitimately output "nothing predicted" quite frequently. It may be that the obscured scene σ specifies too few values, or it may be that the preconditions Q of the rules are very constraining. On the other hand, it is not legitimate for the deduction algorithm to output "nothing predicted" in circumstances when no such valid impediment applies, for then the rules would allow a deduction that is ε-accurate and sound.

To resolve this issue we introduce the following notions.

Definition. For any rule set S with respect to any $\Re, \mathcal{A}, \mathcal{C}$, a *deduction sequence* is a sequence of triples (q_j, π_j, b_j) for $j = 1, \ldots, J$, where $q_j \in S, \pi_j$ is a binding $\{1, \ldots, \alpha_j\} \to \mathcal{A}$ where α_j is the arity of $\mathrm{RHS}(q_j)$, and $b_j \in \{0,1\}$.

Definition. A deduction sequence is *valid* for an input σ if for every j $(1 \leqslant j \leqslant J)$ the following is the case: The values $\mathrm{val}(R,\pi)$ for all pairs (R,π) that are unobscured in the input σ together with the values b_i when taken as the values of $\mathrm{val}(\mathrm{RHS}(q_i),\pi_i)$ for $1 \leqslant i < j$ have the properties that

(i) they satisfy the precondition $Q_j(\pi_j,\sigma)$ of q_j,
(ii) they determine all the expressions in $\mathrm{LHS}(q_j)$, and
(iii) the value of the connective f_j of rule q_j at the specified point equals b_j.

Definition. A deduction algorithm is *complete* for a class of rule sets S with respect to any $\Re, \mathcal{A}, \mathcal{C}$ if and only if the following holds: for any scene $\sigma \in \Pi^{\$}_{\Re,\mathcal{A}}$ there exists a valid deduction sequence terminating with (q,π,b) for some $b \in \{0,1\}$ if and only if the deduction algorithm on input $S, R_i = \mathrm{RHS}(q)$, π, and σ outputs some predicted value from $\{0, 1, 0/1\}$.

If remains to observe:

Proposition. *The acyclic deduction algorithm is complete for acyclic rule sets Γ, and the general deduction algorithm is complete for general rule sets Γ^*.*

Proof. It is easy to show by induction on J that for the general deduction algorithm after the Jth scan all valid deduction sequences of length J have been followed. In other words, if some valid deduction sequence terminates with (q_J, π_J, b_J) then the algorithm will have given a determined value $\{0, 1, 0/1\}$ to node $(\mathrm{RHS}(q_J), \pi_J)$. The main point is that the evaluation process described in Fact 3.1 is monotone, which guarantees that once a node in the evaluation graph is determined it can never become undetermined later. By similar arguments the acyclic deduction algorithm can be seen to be complete. In that case one shows by induction on J that when R_J, is evaluated by the algorithm, all valid deduction sequences terminating in a q_J with that right-hand side have been considered. □

3.4.4. Compound relations and L-expressions

In defining rules we chose, for the sake of simplicity, to describe each quantified expression e_j as depending on a single relation $R_{i_j} \in \Re$. Clearly, we could have a base set $\Re_0$ of relations and have $\Re$ consist of combinations of members of that base set. For example, if we want $\Re$ to have arity 4, we can choose a base set $\Re_0$ of binary relations and define $\Re$ to consist of suitable conjunctions of pairs of these.

The following class of such combinations is noteworthy in applications where it is desirable to reduce the number of bindings that have to be tried in learning or deduction. We define a labelled expression or *L-expression* as an existentially quantified expression.

$$R(x_1, \ldots, x_s) R_1(x_1) R_2(x_2) \cdots R_s(x_s),$$

where R may be an arbitrary compound expression, but $R_1, \ldots, R_s$ are all unary. The *ambiguity* of a scene σ for such an L-expression e with s variables is the number of distinct bindings $\pi : \{x_1, \ldots, x_s\} \to \mathcal{A}$ for which the conjunction

$$R_1(x_1) R_2(x_2) \cdots R_s(x_s)$$

is satisfied for σ. Clearly, if an expression has low ambiguity, say ambiguity 1, with high probability for $\sigma \in D$, then no substantial search over bindings is necessary when recognizing its presence. This seems particularly relevant when modelling biological neural processes, as was previously discussed in [45]. It also provides for more efficient and accurate deduction. For example, suppose that all the expressions occurring in the LHS of a rule set S are L-expressions that have ambiguity one with probability one in D. Then if each of the $\|S\|$ such expressions that occurs is regarded as a separate relation in a new $\Re$ then each of them can have a value 1 for at most one binding, and hence a multiplicative factor of n^α can be saved in the cost of evaluating these expressions. It will also follow that each individual rule may be allowed to be correspondingly less accurate, by a factor of about $n^\alpha |S|/\|S\|$, and still guarantee ε-accuracy of the deduction.

Finally, we observe that the notation described for defining rules can be used to express certain relations of higher arity than the maximum arity of the base relations. We can, for example, have an object, say a_7, represent a set and identify the members of the set to be those a_i for which the relation *member*(a_7, a_i) holds. If we have a binary relation *adjacent* (x_j, x_k) then we can express the relation *completely-connected*(x_i), to simulate a relation of arity up to $n - 1$, as follows:

$$\forall x_1 \in \mathcal{A}\left[\forall x_2 \forall x_3 \big(\neg \mathit{member}(x_1, x_2) \vee \neg \mathit{member}(x_1, x_3) \vee \mathit{adjacent}(x_2, x_3)\big) \equiv \mathit{completely\text{-}connected}(x_1)\right].$$

If the relations *member*( ) and *adjacent*( ) are predefined, and the LHS of this rule is a member of the allowed set of compound relations, then such a rule for *completely-connected*( ) can be learned from examples, or evaluated.

4. Related work

The framework described here has relationships with work in many other areas. Relational rules and their evaluation are central, for example, to both databases and artificial intelligence, while learning has been studied in numerous settings. Our logic is distinguished by the close way in which it integrates learning and reasoning, and by the insistence on having the measures of both computational complexity and of accuracy bounded by polynomial functions.

Among the numerous alternative approaches that are relevant to ours we briefly mention just three.

Probabilistic logics have been defined to extend predicate calculus to probabilistic phenomena [6]. Some treat the underlying distribution as one over the world, while others impose it on a space of beliefs (see [12]). PAC-semantics has aspects of both. The underlying distribution is one over the world, but the degree of belief in any rule that the system is justified to have at any time may also be quantified, for example, by a confidence parameter δ.

Inductive Logic Programming is concerned with learning rules in predicate calculus, with the standard semantics, often with the additional constraint of a background logical theory. The limits of polynomial time learnability have been explored [8,21,30]. Some systems exist for learning such rules [34]. A discussion of some relationships between the two approaches can be found in [20].

The *Learning to Reason* framework [18,19] integrates learning and reasoning in a similar spirit to our approach. Its most basic form is oriented towards maintaining information as a set of examples, and answering each query by processing these examples anew. Robust logics are based on rules. This allows a system to accept a rule as input and hence to benefit from the experience of others, while still permitting it to evaluate or fine-tune the rule on the basis of further examples and its own experience. Clearly systems based on robust logics may additionally memorize individual examples, and thereby enjoy whatever added benefits may be so gained.

5. Conclusion

We have described a formal system for representing knowledge that has a well defined semantics and an efficient, sound and complete deduction procedure for instances. It also has an efficient learning procedure for learning the representation of rules one at a time, if an appropriate class of connectives that is efficiently learnable is used. The class of

linear threshold functions appears to be a particular good choice since it offers, in addition, attribute-efficient and error-resilient learning.

Clearly there are many directions in which one might attempt extensions. For example, we have restricted the discussion of reasoning here to deductions about single instances. It would be interesting to see what can be said about the deduction of new rules. Also, since PAC-semantics is based on probability theory, all the methods of probabilistic inference can be brought to bear (e.g., [32]).

There also remain questions of how such a logic can be applied pragmatically when building computer systems that perform computations of a cognitive nature. The discussion in [47] addresses some of these questions.

References

[1] J.R. Anderson, Rules of the Mind, Erlbaum, Hillsdale, NJ, 1993.
[2] D. Angluin, P. Laird, Learning from noisy examples, Machine Learning 2 (4) (1988) 343–370.
[3] Aristotle, Prior Analytics, Book II, Part 23.
[4] A. Blum, A. Frieze, R. Kannan, S. Vempala, A polynomial time algorithm for learning noisy linear threshold functions, in: Proc. 37th IEEE Annual Symposium on Foundations of Computer Science (FOCS-96), 1996, pp. 330–338.
[5] T. Bylander, Learning linear threshold functions in the presence of classification noise, in: Proc. 7th ACM Conference on Computational Learning Theory, 1994, pp. 340–347.
[6] R. Carnap, Logical Foundations of Probability, University of Chicago Press, Chicago, IL, 1950.
[7] E. Cohen, Learning noisy perceptrons by a perceptron in polynomial time, in: Proc. 38th IEEE Symposium on Foundation of Computer Science, 1997, pp. 514–523.
[8] W.W. Cohen, C.D. Page, Polynomial learnability and inductive logic programming: Methods and results, New Generation Computing 13 (314) (1995) 369–409.
[9] A. Ehrenfeucht, D. Haussler, M. Kearns, L. Valiant, A general lower bound on the number of examples needed for learning, Inform. and Comput. 82 (3) (1989) 247–266.
[10] M.L. Ginsberg, Readings in Nonmonotonic Reasoning, Morgan Kaufmann, Los Altos, CA, 1989.
[11] A.R. Golding, D. Roth, Applying Winnow to context-sensitive spelling correction, in: Proc. 13th Internat. Conference on Machine Learning, Bari, Italy, Morgan Kaufmann, San Mateo, CA, 1996, pp. 182–190.
[12] J.Y. Halpern, An analysis of first-order logics of probability, Artificial Intelligence 46 (1990) 311–350.
[13] D. Haussler, Quantifying inductive bias: AI learning algorithms and Valiant's learning framework, Artificial Intelligence 36 (1988) 177–221.
[14] M.J. Kearns, Efficient noise-tolerant learning from statistical queries, in: Proc. 25th ACM Symposium on Theory of Computing, ACM Press, New York, 1993, pp. 392–401.
[15] M. Kearns, M. Li, Learning in the presence of malicious errors, SIAM J. Comput. 22 (4) (1993) 807–837.
[16] M.J. Kearns, U.V. Vazirani, An Introduction to Computational Learning Theory, MIT Press, Cambridge, MA, 1994.
[17] R. Khardon, Learning to take actions, in: Proc. AAAI-96, Portland, OR, 1996, pp. 787–792.
[18] R. Khardon, D. Roth, Learning to reason with a restricted view, in: Proc. 8th ACM Conference on Computational Learning Theory, 1995, pp. 301–310.
[19] R. Khardon, D. Roth, Learning to reason, J. ACM 44 (5) (1997) 697–772.
[20] R. Khardon, D. Roth, L.G. Valiant, Relational learning for NLP using linear threshold elements, in: Proc. IJCAI-99, Stockholm, Sweden, Morgan Kaufmann, San Mateo, CA, 1999, pp. 911–917.
[21] J.-U. Kietz, S. Dzeroski, Inductive logic programming and learnability, SIGART Bulletin 5 (1) (1994) 22–32.
[22] J. Kivinen, M.K. Warmuth, P. Auer, The Perceptron algorithm versus Winnow: Linear versus logarithmic mistake bounds when few input variables are relevant, in: Proc. 8th ACM Conference on Computational Learning Theory, 1995, pp. 289–296; also: Artificial Intelligence 97 (1997) 325–343.
[23] D. Knuth, The Art of Computer Programming, Vol. 3, Addison Wesley, Reading, MA, 1973.

[24] P. Langley, D. Klahr, R. Neches, Production System Models of Learning and Development, MIT Press, Cambridge, MA, 1987.
[25] D.B. Lenat et al., CYC: Toward programs with common sense, Comm. ACM 33 (8) (1990) 30–49.
[26] N. Littlestone, Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm, Machine Learning 2 (1988) 285–318.
[27] N. Littlestone, From on-line to batch learning, in: Proc. 2nd Workshop on Computational Learning Theory, 1989, pp. 269–284.
[28] J. McCarthy, Programs with commonsense, in: Proc. Teddington Conference on the Mechanization of Thought Processes, London, 1959. HMSO.
[29] M. Minsky, S. Papert, Perceptrons, MIT Press, Cambridge, MA, 1969.
[30] S. Muggleton, L. De Raedt, Inductive logic programming: Theory and methods, J. Logic Programming 19 (1994) 629–679.
[31] A. Newell, H.A. Simon, Human Problem Solving, Prentice-Hall, Englewood Cliffs, NJ, 1972.
[32] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, Los Altos, CA, 1988.
[33] L. Pitt, L.G. Valiant, Computational limits on learning from examples, J. ACM 35 (1988) 965–984.
[34] J.R. Quinlan, Learning logical definitions from relations, Machine Learning 5 (1990) 239–266.
[35] R.L. Rivest, Learning decision lists, Machine Learning 2 (3) (1987) 229–246.
[36] F. Rosenblatt, Principles of Neurodynamics, Spartan, New York, 1962.
[37] D. Roth, Learning to reason: The non-monotonic case, in: Proc. IJCAI-95, Montreal, Quebec, 1995, pp. 1178–1184.
[38] D. Roth, A connectionist framework for reasoning: Reasoning with examples, in: Proc. AAAI-96, Portland, OR, 1996, pp. 1256–1261.
[39] S. Russell, P. Norvig, Artificial Intelligence, Prentice-Hall, Upper Saddle River, NJ, 1995.
[40] D. Schuurmans, R. Greiner, Learning default concepts, in: Proc. 10th Canadian Conference on Artificial Intelligence, CSCSI-96, Toronto, Ont., 1994, pp. 99–106.
[41] A.M. Turing, Computing machinery and intelligence, Mind 59 (1950) 433–460; Reprinted in: D.C. Ince (Ed.), Collected Works of A.M. Turing: Mechanical Intelligence, North-Holland, Amsterdam, 1992.
[42] A.M. Turing, Solvable and unsolvable problems, Science News 31 (1954) 7–23; Reprinted in: D.C. Ince (Ed.), Collected Works of A.M. Turing: Mechanical Intelligence, North-Holland, Amsterdam, 1992.
[43] J.D. Ullman, Principles of Database and Knowledge-Base Systems, Computer Science Press, 1989.
[44] L.G. Valiant, A theory of the learnable, Comm. ACM 27 (11) (1984) 1134–1142.
[45] L.G. Valiant, Circuits of the Mind, Oxford University Press, Oxford, 1994.
[46] L.G. Valiant, Rationality, in: Proc. 8th Annual Conference on Computational Learning Theory, ACM Press, New York, 1995, pp. 3–14.
[47] L.G. Valiant, A neuroidal architecture for cognitive computation, in: Lecture Notes in Computer Science, Vol. 1443, Springer, Berlin, 1998, pp. 642–669; also: J. ACM, to appear.
[48] L.G. Valiant, Projection learning, Machine Learning 37 (2) (1999) 115–130.