In: *Proceedings of CoNLL-2000 and LLL-2000*, pages 176–183, Lisbon, Portugal, 2000.

Learning from a Substructural Perspective

Pieter Adriaans and **Erik de Haas**
Syllogic,
P.O. Box 2729, 3800GG Amersfoort, The Netherlands,
and
University of Amsterdam, Fac. of Mathematics, Computer Science, Physics and Astronomy,
Plantage Muidergracht 24, 1018TV Amsterdam, The Netherlands
pieter.adriaans@ps.net, erik@propersolution.nl

Abstract

In this paper we study learning from a logical perspective. We show that there is a strong relationship between a learning strategy, its formal learning framework and its logical representational theory. This relationship enables one to translate learnability results from one theory to another. Moreover if we go from a classical logic theory to a substructural logic theory, we can transform learnability results of logical concepts to results for string languages. In this paper we will demonstrate such a translation by transforming the Valiant learnability result for boolean concepts to a learnability result for a class of string pattern languages.

1 Introduction

There is a strong relation between a learning strategy, its formal learning framework and its representational theory. Such a representational theory typically is (equivalent to) a logic. As an example for this strong relationship assume that the implication $A \rightarrow B$ is a given fact, and you observe A; then you can deduce B, which means that you can learn B from A based on the underlying representational theory. The learning strategy is very tightly connected to its underlying logic. Continuing the above example, suppose you observe $\neg B$. In a representational theory based on classical logic you may deduce $\neg A$ given the fact $A \rightarrow B$. In intuitionistic logic however, this deduction is not valid. This example shows that the character of the representational theory is essential for your learning strategy, in terms of what can be learned from the facts and examples.

In the science of the representational theories, i.e. logic, it is a common approach to connect different representational theories, and transform results of one representational theory to results in an other representational theory. Interesting is now whether we can transform learnability results of learning strategies within one representational theory to others. Observe that to get from a first order calculus to a string calculus one needs to eliminate structural rules from the calculus. Imagine now that we do the same transformation to the learning strategies, we would come up with a learning strategy for the substructural string calculus starting from a learning strategy for the full first order calculus.

The observation that learning categorial grammars translates to the task of learning derivations in a substructural logic theory motivates a research program that investigates learning strategies from a logical point of view (Adriaans and de Haas, 1999). Many domains for learning tasks can be embedded in a formal learning framework based on a logical representational theory. In Adriaans and de Haas (1999) we presented two examples of substructural logics, that were suitable representational theories for different learning tasks; The first example was the Lambek calculus for learning categorial grammars, the second example dealt with a substructural logic that was designed to study modern Object Oriented modeling languages like UML (OMG, 1997), (Fowler, 1997). In the first case the representation theory is first order logic without structural rules, the formal learning theory from a logical point of view is inductive substructural logic programming and an example of a learning strategy in this framework is EMILE, a learning algorithm that learns categorial grammars (Adriaans, 1992).

In this paper we concentrate on the transformation of classical logic to substructural logic and show that Valiant's proof of PAC-learnability of boolean concepts can be transformed to a PAC learnability proof for learning a class of finite languages. We discuss the extension of this learnability approach to the full range of substructural logics. Our strategy in exploring the concept of learning is to look at the logical structure of a learning algorithm, and by this reveal the inner working of the learning strategy.

In Valiant (1984) the principle of Probably Approximately Correct learning (PAC learning) was introduced. There it has been shown that k-CNF (k-length Conjunctive Normal Form) boolean concepts can be learned efficiently in the model of PAC learning. For the proof that shows that these boolean concepts can be learned efficiently Valiant presents a learning algorithm and shows by probabilistic arguments that boolean concept can be PAC learned in polynomial time. In this paper we investigate the logical mechanism behind the learning algorithm. By revealing the logical mechanism behind this learning algorithm we are able to study PAC learnability of various other logics in the substructural landscape of first order propositional logic.

In this paper we will first briefly introduce substructural logic in section 2. Consequently we will reconstruct in section 3 Valiant's result on learnability of boolean concepts in terms of logic. Then in section 4 we will show that the learnability result of Valiant for k-CNF boolean concepts can be transformed to a learnability result for a grammar of string patterns denoted by a substructural variant of the k-CNF formulas. We will conclude this paper with a discussion an indicate how this result could be extended to learnability results for categorial grammars.

2 Substructural logic

In Gentzen style sequential formalisms a *substructural logic* shows itself by the absence of (some of) the so-called structural rules. Examples of such logics are relevance logic (Dunn, 1986), linear logic (Girard, 1987) and BCK logic (Grishin, 1974). Notable is the substructural behavior of categorial logic, which in its prototype form is the Lambek calculus. Categorial logics are motivated by its use as grammar for natural languages. The absence of the structural rules degrades the abstraction of *sets* in the semantic domain to *strings*, where elements in a string have position and arity, while they do not have that in a set. As we will see further on in this paper the elimination of the structural rules in the learning context of the boolean concepts will transform the learning framework from *sets* of valuated variables to *strings* of valuated variables.

Example 2.1 *In a domain of sets the following 'expressions' are equivalent, while they are not in the domain of strings:*

$$a, a, b, a \approx a, b, b$$

In a calculus with all the structural rules the features 'position' and 'arity' are irrelevant in the semantic domain, because aggregates that differ in these features can be proved equivalent with the structural rules. To see this observe that the left side of the above equation can be transformed to the right side by performing the following operation:

a, a, b, a
  contract a, a *in first two positions to* a
a, b, a
  exchange b, a *in last to positions to* a, b
a, a, b
  contract *again* a, a *in first two positions to* a
a, b
  weaken *expression* b *in last position to* b, b
a, b, b

In figure 2 we list the axiomatics of the first order propositional sequent calculus[1], with the axioms , the cut rule, rules for the connectives and the structural rules for exchange, weakening and contraction.

3 PAC Boolean concept learning revisited

In this section we describe the principle of Probably Approximately Correct Learning (PAC learning) of Boolean concepts. We will reveal

[1]Note that in the variant we use here we have a special case of the R$\wedge$ rule.

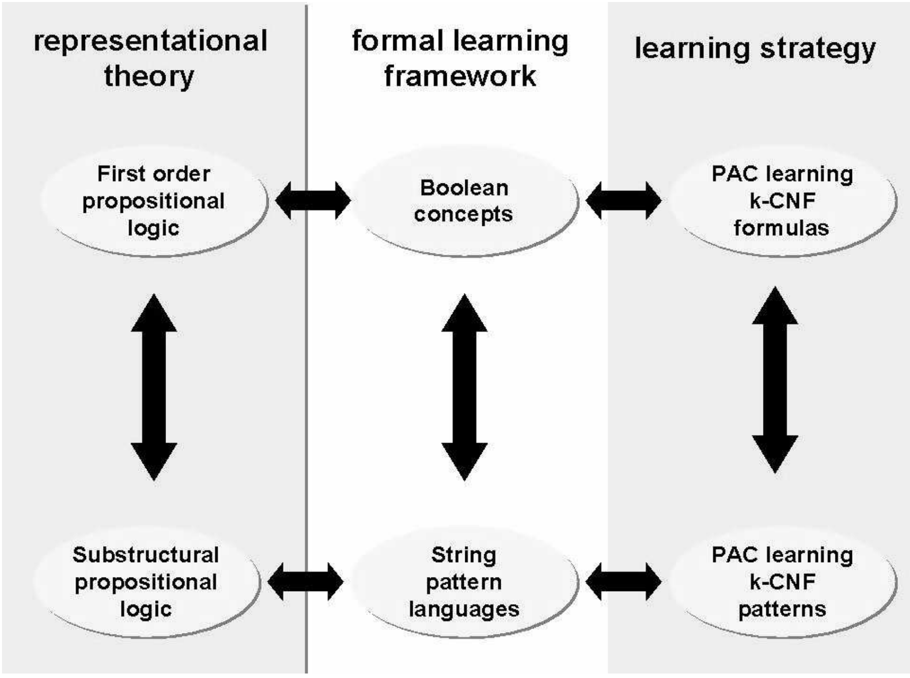

Figure 1: Relation between learning strategy, learning framework and representational theory

$$
\begin{array}{llll}
(\mathrm{Ax}) & A \Rightarrow A & (\mathrm{Cut}) & \dfrac{\Gamma \Rightarrow A, \Delta \quad \Gamma', A, \Rightarrow \Delta}{\Gamma', \Gamma \Rightarrow \Delta', \Delta} \\
(\mathrm{L}\wedge) & \dfrac{\Gamma, A, B \Rightarrow \Delta}{\Gamma, A \wedge B \Rightarrow \Delta} & (\mathrm{R}\wedge) & \dfrac{\Gamma \Rightarrow A, \Delta \quad \Gamma' \Rightarrow B, \Delta}{\Gamma, \Gamma' \Rightarrow A \wedge B, \Delta} \\
(\mathrm{L}\vee) & \dfrac{\Gamma, A \Rightarrow \Delta \quad \Gamma, B \Rightarrow \Delta}{\Gamma, A \vee B \Rightarrow \Delta} & (\mathrm{R}\vee) & \dfrac{\Gamma \Rightarrow A, \Delta}{\Gamma \Rightarrow A \vee B, \Delta} \quad \dfrac{\Gamma \Rightarrow B, \Delta}{\Gamma \Rightarrow A \vee B, \Delta}
\end{array}
$$

$$
\begin{array}{ll}
(\mathrm{Ex}) & \dfrac{\Gamma, A \wedge B, \Gamma' \Rightarrow \Delta}{\Gamma, B \wedge A, \Gamma' \Rightarrow \Delta} \\
(\mathrm{Weak}) & \dfrac{\Gamma \Rightarrow \Delta}{\Gamma, A \Rightarrow \Delta} \\
(\mathrm{Contr}) & \dfrac{\Gamma, A, A \Rightarrow \Delta}{\Gamma, A \Rightarrow \Delta}
\end{array}
$$

Figure 2: First order propositional sequent calculus

the logical deduction process behind the learning algorithm.

Consider the sample space for boolean concepts. An example is a vector denoting the truth (presence,1) or falsehood (absence,0) of propositional variables. Such an example vector can be described by a formula consisting of the conjunction of all propositional variables or negations of propositional variables, depending on the fact whether there is a 1 or a 0 in the position of the propositional variable name in the vector. A collection of vectors, i.e. a concept, in its turn can be denoted by a formula too, being the disjunction of all the formula's of the vectors.

Example 3.1 *Let universe* $\mathsf{U} = \{a, b\}$ *and let concept* $f = \{(0,1)\}$*, then the following formula exactly describes* f*:*

$$\overline{a} \wedge b$$

A little more extensive: Let universe $\mathsf{U}' = \{a, b, c\}$ *and let concept* $f' = \{(0,0,0),(0,0,1),(0,1,1),(1,1,1)\}$ *Then the following formula exactly describes* f' *(with a clear translation):*

$$(\overline{a} \wedge \overline{b} \wedge \overline{c}) \vee (\overline{a} \wedge \overline{b} \wedge c) \vee (\overline{a} \wedge b \wedge c) \vee (a \wedge b \wedge c)$$

Note that these formulas are in Disjunctive normal form (DNF).

An interesting observation now is that the learning algorithm of Valiant that learns k-CNF formulas actually is trying to prove the equivalence between a DNF formula and a k-CNF formula.

Example 3.2 *Let universe* $\mathsf{U} = \{a, b\}$ *and let concept* $f = \{(0,1)\}$*, then the following sequent should be 'learned' by a 2-CNF learning algorithm*[2]*:*

$$\overline{a} \wedge b \Leftrightarrow (a \vee b) \wedge (\overline{a} \vee b) \wedge (\overline{a} \vee \overline{b})$$

A little more extensive: Let $\mathsf{U}' = \{a, b, c\}$ *and let concept* $f' = \{(0,0,0),(0,0,1),(0,1,1),(1,1,1)\}$ *Then the following sequent should be 'learned' by a 2-CNF learning algorithm:*

$$(\overline{a} \wedge \overline{b} \wedge \overline{c}) \vee (\overline{a} \wedge \overline{b} \wedge c) \vee (\overline{a} \wedge b \wedge c) \vee (a \wedge b \wedge c)$$
$$\Leftrightarrow (\overline{a} \vee \overline{b}) \wedge (\overline{a} \vee b) \wedge (a \vee b)$$

The above observation says in logical terms that the learning algorithm needs to implement an inductive procedure to find this desired proof and the concluding concept description (2-CNF formula) from examples. In the search space for this proof the learning algorithm can use the axioms and rules from the representational theory. In the framework of boolean concept learning this means that the learning algorithm may use all the rules and axioms from the representational theory of classical propositional logic.

Example 3.3 *Let* $\mathsf{U} = \{a, b\}$ *and let concept* $f = \{(0,1)\}$ *and assume* f *can be represented by a 2-CNF formula. to learn the 2-CNF description of concept* f *the learning algorithm needs to find the proof for a sequent starting from the DNF formula* $\overline{a} \wedge b$ *to a 2-CNF formula and vice versa* ($\Leftrightarrow$) *and to do so it may use all the rules and axioms from the first order propositional calculus including the structural rules. The proof for one side of such a sequent is spelled out in figure 3.*

[2] i.e. an algorithm that can learn 2-CNF boolean concepts.

In general an inductive logic programming algorithm for the underlying representational theory can do the job of learning the concept; i.e. from the examples (DNF formulas) one can induce possible sequents, targeting on a 2-CNF sequent on the righthand side. The learning algorithm we present here is more specific and simply shows that an efficient algorithm for the proof search exists.

The steps:

1. `Form the collection` G `of all 2-CNF clauses` $(p \vee q)$
2. `do` l `times`
   - (a) `pick an example` $a_1 \wedge \cdots \wedge a_m$
   - (b) `form the collection of all 2-CNF clauses deducible from` $a_1 \wedge \cdots \wedge a_m$ `and intersect this collection with` G `resulting in a new` G

Correctness proof (outline): By (Ax), ($R\vee$), (Weak), ($L\wedge$) and (Ex) we can proof that for any conjunction (i.e. example vector) $a_1 \wedge \cdots \wedge a_m$ we have for all $1 \leq i \leq m$ and any b a clause of a 2-CNF in which a_i occurs with b , hence having all clauses deducible from the vector proven individually enabling one to form the collection of all clauses deducible from a vector; i.e.

$$a_1 \wedge \cdots \wedge a_m \Rightarrow a_i \vee b$$
$$a_1 \wedge \cdots \wedge a_m \Rightarrow b \vee a_i$$

By ($R\wedge$) and (Contr) we can proof the conjunction of an arbitrary subset of all the clauses deducible from the vector, in particular all those clauses that happen to be common to all the vectors for each individual vector we have seen so far, hence proving the 2-CNF for every individual vector; i.e.

$$a_1 \wedge \cdots \wedge a_m \Rightarrow \text{clause}_1 \wedge \cdots \wedge \text{clause}_p$$

$$
\begin{prooftree}
\AxiomC{$\overline{a} \Rightarrow \overline{a}$ (Ax)}
\RightLabel{(R$\vee$)}
\UnaryInfC{$\overline{a} \Rightarrow \overline{a} \vee \overline{b}$}
\RightLabel{(Weak)}
\UnaryInfC{$\overline{a}, b \Rightarrow \overline{a} \vee \overline{b}$}
\RightLabel{(L$\wedge$)}
\UnaryInfC{$\overline{a} \wedge b \Rightarrow \overline{a} \vee \overline{b}$}
\AxiomC{$b \Rightarrow b$ (Ax)}
\RightLabel{(R$\vee$)}
\UnaryInfC{$b \Rightarrow \overline{a} \vee b$}
\RightLabel{(Weak)}
\UnaryInfC{$b, \overline{a} \Rightarrow \overline{a} \vee b$}
\RightLabel{(L$\wedge$)}
\UnaryInfC{$b \wedge \overline{a} \Rightarrow \overline{a} \vee b$}
\RightLabel{(Ex)}
\UnaryInfC{$\overline{a} \wedge b \Rightarrow \overline{a} \vee b$}
\AxiomC{$b \Rightarrow b$ (Ax)}
\RightLabel{(R$\vee$)}
\UnaryInfC{$b \Rightarrow a \vee b$}
\RightLabel{(Weak)}
\UnaryInfC{$b, \overline{a} \Rightarrow a \vee b$}
\RightLabel{(L$\wedge$)}
\UnaryInfC{$b \wedge \overline{a} \Rightarrow a \vee b$}
\RightLabel{(Ex)}
\UnaryInfC{$\overline{a} \wedge b \Rightarrow a \vee b$}
\RightLabel{(R$\wedge$)}
\BinaryInfC{$(\overline{a} \wedge b), (\overline{a} \wedge b) \Rightarrow (\overline{a} \vee b) \wedge (a \vee b)$}
\RightLabel{(R$\wedge$)}
\BinaryInfC{$(\overline{a} \wedge b), (\overline{a} \wedge b), (\overline{a} \wedge b) \Rightarrow (\overline{a} \vee \overline{b}) \wedge (\overline{a} \vee b) \wedge (a \vee b)$}
\RightLabel{(Contr)}
\UnaryInfC{$(\overline{a} \wedge b), (\overline{a} \wedge b) \Rightarrow (\overline{a} \vee \overline{b}) \wedge (\overline{a} \vee b) \wedge (a \vee b)$}
\RightLabel{(Contr)}
\UnaryInfC{$(\overline{a} \wedge b) \Rightarrow (\overline{a} \vee \overline{b}) \wedge (\overline{a} \vee b) \wedge (a \vee b)$}
\end{prooftree}
$$

Figure 3: Proof to be found for boolean concept learning

Now by (LV) we can prove the complete DNF to 2-CNF sequent; i.e.

$$\text{vector}_1 \vee \cdots \vee \text{vector}_l \Rightarrow \text{clause}_1 \wedge \cdots \wedge \text{clause}_p$$

It is easy to see that for the above algorithm the same complexity analysis holds as for the Valiant algorithm, because we have the same progression in l steps, an the individual steps have constant overhead.

4 PAC learning substructural logic

When we transform the representational theory of the boolean concept learning framework to a substructural logic, we do the following:

- eliminate the structural rules from the calculus of first order propositional logic

When we want to translate the learnability result of k-CNF expressible boolean concepts we need to do the same with the formal learning framework and the strategy (algorithm). In other words:

- the learning framework will contain concepts that are sensitive to the features which were before abstracted by the structural rules ('position' and 'arity')
- the learning algorithm from above is no longer allowed to use the structural rules in its inductive steps.

Below we present a learning algorithm for the substructural logic representational theory. Suppose again the universe $\mathsf{U} = \{a_1, \ldots, a_n\}$, and the concept f is a CNF expressible concept for vectors of length m.

1. start with m empty clauses (i.e. disjunction of zero literals) $\text{clause}_1, \ldots, \text{clause}_m$
2. do l times
   (a) pick an example $a_1 \wedge \cdots \wedge a_m$
   (b) for all $1 \leq i \leq m$ add a_i to clause_i if a_i does not occur in clause_i.

Correctness proof (outline): By (Ax) and $(R\vee)$ we can proof for any a_i that the sequent $a_i \Rightarrow \text{clause}_i$ for any clause_i containing a_i as one of its disjuncts, especially for a clause_i containing next to a_i all the a'_i from the former examples. Then by $(R\wedge)$ and $(L\wedge)$ we can position all the vectors and clauses in the right-hand position; i.e.

$$a_1 \wedge \cdots \wedge a_m \Rightarrow \text{clause}_1 \wedge \cdots \wedge \text{clause}_m$$

Hence justifying the adding of the literal a_i of a vector in clause_i. Now (LV) completes the sequent for all the example vectors; i.e.

$$(a_1 \wedge \cdots \wedge a_m) \vee (a'_1 \wedge \cdots \wedge a'_m) \vee \cdots$$
$$\Rightarrow \text{clause}_1 \wedge \cdots \wedge \text{clause}_m$$

For the algorithmic complexity in terms of PAC learning, suppose we want present examples of concept f and that the algorithm learned concept f' in l steps. Concept f' then describes a *subset* of concept f because on every position in the CNF formula contains a subset of the allowed variables; i.e. those variables that have encountered in the examples[3].

[3] note that the CNF formula's can only describe particular sets of n-strings; namely those sets that are complete for varying symbols locally on the different positions in the string.

$$
\dfrac{\cdots \quad \cdots \quad \dfrac{\dfrac{\dfrac{\overline{a} \Rightarrow \overline{a}\ (\mathrm{Ax})}{\overline{a} \Rightarrow \overline{a} \vee \overline{b}}\,(\mathrm{R}\vee) \quad \dfrac{\dfrac{\dfrac{b \Rightarrow b\ (\mathrm{Ax})}{b \Rightarrow \overline{a} \vee b}\,(\mathrm{R}\vee) \quad \dfrac{b \Rightarrow b\ (\mathrm{Ax})}{b \Rightarrow a \vee b}\,(\mathrm{R}\vee)}{b, b \Rightarrow (\overline{a}\vee) \wedge (a \vee b)}\,(\mathrm{R}\wedge)}{b \wedge b \Rightarrow (\overline{a} \vee b) \wedge (a \vee b)}\,(\mathrm{L}\wedge)}{\overline{a}, b \wedge b \Rightarrow (\overline{a} \vee \overline{b}) \wedge (\overline{a} \vee b) \wedge (a \vee b)}\,(\mathrm{R}\wedge)}{\overline{a} \wedge b \wedge b \Rightarrow (\overline{a} \vee \overline{b}) \wedge (\overline{a} \vee b) \wedge (a \vee b)}\,(\mathrm{L}\wedge) \quad \cdots}{\begin{array}{c}(\overline{a} \wedge \overline{a} \wedge a) \vee (\overline{a} \wedge \overline{a} \wedge b) \vee (\overline{a} \wedge b \wedge a) \vee (\overline{a} \wedge b \wedge b) \vee (\overline{b} \wedge \overline{a} \wedge a) \\ \vee (\overline{b} \wedge \overline{a} \wedge b) \vee (\overline{b} \wedge b \wedge a) \vee (\overline{b} \wedge b \wedge b) \Rightarrow (\overline{a} \vee \overline{b}) \wedge (\overline{a} \vee b) \wedge (a \vee b)\end{array}}\,(\mathrm{L}\vee)
$$

Figure 4: Proof to be found for string pattern learning

Now let $\epsilon = P(f \Delta f')$ be the error then again $\delta = (1-e)^m$ is the confidence parameter as we have m positions in the string. By the same argument as for the Valiant algorithm we may conclude that ϵ and δ decrease exponentially in the number of examples l, meaning that we have an efficient polynomial time learning algorithm for arbitrary ϵ and δ.

5 Discussion

We showed that the learnability result of Valiant for learning boolean concepts can be transformed to a learnability result for pattern languages by looking at the transformation of the underlying representational theories; i.e. looking at the transformation from classical first order propositional logic (underlying the boolean concepts) to substructural first order propositional logic (underlying the pattern languages). An interesting extension would be to look at the substructural concept language that includes *implication* (instead of the CNF formula's only). A language that allows implication coincides with the full Lambek calculus, and a learning algorithm and learnability result for this framework amounts to results for all languages that can be described by context free grammars. This is subject to future research.

References

P. Adriaans and E. de Haas. 1999. Grammar induction as substructural inductive logic programming. In *Proceedings of the workshop on Learning Language in Logic (LLL99)*, pages 117–126, Bled, Slovenia, jun.

P. Adriaans. 1992. *Language Learning from a Categorial Perspective*. Ph.D. thesis, Universiteit van Amsterdam. Academisch proefschrift.

J. Dunn. 1986. Relevance logic and entailment. In F. Guenthner D. Gabbay, editor, *Handbook of Philosophical Logic III*, pages 117–224. D. Reidel Publishing Company.

M. Fowler. 1997. *UML Distilled: Applying the Standard Object Modeling Language*. Addison Wesley Longman.

J.-Y. Girard. 1987. Linear logic. *Theoretical Computer Science*, 50:1–102.

V.N. Grishin. 1974. A non-standard logic, and its applications to set theory. In *Studies in formalized languages and nonclassical logics*, pages 135–171. Nauka.

Object Management Group OMG. 1997. Uml 1.1 specification. OMG documents ad970802-ad0809.

L.G. Valiant. 1984. Theory of the learnable. *Comm. of the ACM*, 27:1134–1142.

Addendum: PAC learning

The model of PAC learning arises from the work of Valiant (Valiant, 1984). In this model of learning it is assumed that we have a *sample space* U^* of vectors over an alphabet U, where each position in a vector denotes the presence (1) or absence (0) of a symbol $a \in \mathsf{U}$ in the sample vector. A *concept* f is a subset of vectors from the sample space U^*.

Example 5.1 *Let $\mathsf{U} = \{a, b\}$ be an alphabet, then the following table describes the sample space U^* over U:*

a	b
0	*0*
0	*1*
1	*0*
1	*1*

an example of a concept is $f := \{(0,1)\}$ and an other example is $g := \{(0,0),(0,1),(1,1)\}$.

A concept can be learned by an algorithm by giving this algorithm positive and/or negative examples of the target concept to be learned. An algorithm *efficiently learns* a concept if this algorithm produces a description of this concept in polynomial time. Informally a concept is *PAC (Probably Approximately Correct) learned* if the algorithm produces a description of a concept that is by approximation the same as the target concept from which examples are feeded into the algorithm. A collection of concepts constitutes to a *concept class*. A concept class can be (PAC) learned if all the concepts in the concept class can be (PAC) learned.

Definition 5.2 *(PAC Learnable) Let F be a concept class, δ $(0 \leq \delta \leq 1)$ a confidence parameter, ϵ $(0 \leq \epsilon \leq 1)$ an error parameter. A concept class F is* PAC learnable *if for all targets $f \in F$ and all probability distributions P on the sample space U^* the learning algorithm A outputs a concept $g \in F$ such that with probability $(1-\delta)$ it holds that we have a chance on an error with $P(f \Delta g) \leq \epsilon$ (where $f \Delta g = (f-g) \cup (g-f)$)*

We are especially interested in concept classes that are defined by some formalism (language). In other words a language can describe come collection of concepts. An example of such a language is the language of boolean formulas. A boolean formula describes a concept that consists of all the vectors over the alphabet of propositional variable names that satisfy the formula. These concepts are called *boolean concepts.*

Example 5.3 *Let $\mathsf{U} := \{a, b\}$ be an alphabet of propositional variable names. Then the formula $\overline{a} \wedge b$ describes the concept $f := \{(0,1)\}$ of the sample space U^*; and the formula $\overline{a} \vee b$ describes the concept $g := \{(0,0),(0,1),(1,1)\}$.*

In Valiant (1984) Valiant proves that the language of k-CNF boolean formula's can be efficiently PAC learned. This means that for an arbitrary k the concept class defined by the language of k-CNF formula's can be PAC learned by an algorithm in a polynomial number of steps. Below we briefly recapitulate this result.

Definition 5.4 *(Boolean concept languages) Let U be a set of propositional variable names, then the language L of boolean formulas is defined by:*

$$L := \mathsf{U} | L \vee L | L \wedge L | \overline{L}$$

A literal *is a propositional variable or a negation of a propositional variable; i.e.*

$$\mathsf{LIT} := \mathsf{U} | \overline{\mathsf{U}}$$

A conjunction of a collection of formulas C is a finite sequence of formulas from C connected by the binary connective $\wedge$; i.e.

$$\mathsf{CON}(C) := C | \mathsf{CON}(C) \wedge C$$

A disjunction of a collection of formulas C is a finite sequence of formulas from C connected by the binary connective $\vee$; i.e.

$$\mathsf{DIS}(C) := C | \mathsf{DIS}(C) \vee C$$

A formula is a CNF formula (Conjunctive Normal Form) if the formula is a conjunction of disjunctions of literals. A formula is a k-CNF formula if all the disjuctions in the formula are of length k. A formula is a DNF formula (Disjunctive Normal Form) if the formula is a disjunction of conjunctions of literals.

Theorem 5.5 *(Valiant (1984)) The classes of k-CNF boolean concept languages are PAC learnable in polynomial time.*

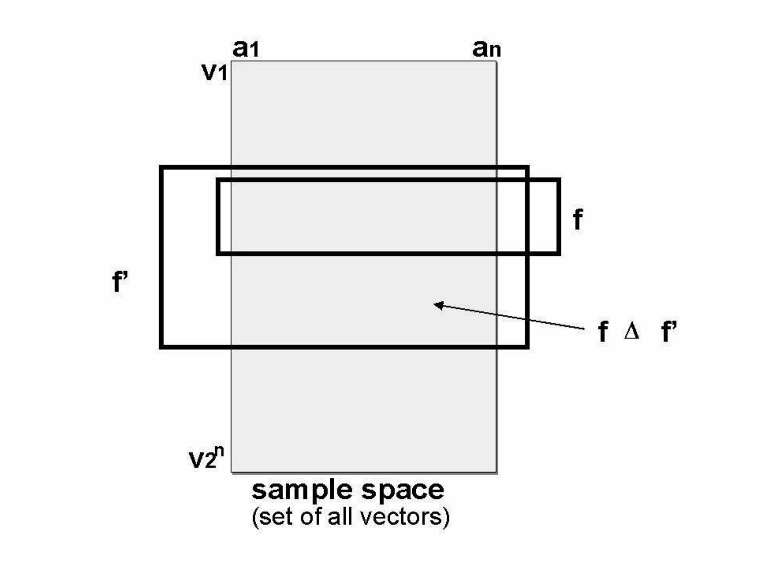

Figure 5: Valiant's proof for boolean concept learning

Proof (outline): Let $\mathsf{U} := \{a_1, \ldots, a_n\} (n \in \mathcal{N})$ be a alphabet and let concept f be a set of vectors $V := \{v_1, \ldots, v_m\} (m \leq n)$ over U^*, which is equivalent to the k-CNF formula A.

Let P be an arbitrary probability distribution over concept f such that $\Sigma_{v_i \in f} P(v_i) = 1$; i.e. $P(f) = 1$. Examples picked using the distribution based on P will be feeded into the following learning algorithm:

- `Form the collection` $G := \{c_1, \ldots, c_{n^k}\}$ `of all the clauses (disjunctions of literals) of length` k.
- `do` l `times`
  - $v :=$ `pick-an-example`
  - `for each` c_i `in` G
    - `delete` c_i `from` G `if` $v \not\Rightarrow c_i$

Now suppose that the algorithm learned concept f' from l examples (l taken from the algorithm). The concept f' now is a concept that is a subset of f, because it may not have seen enough examples to eliminate all the clauses that are in conflict with f; i.e. there are still clauses in f' restricting this concept in the conjunction of clauses, while it is disqualified by a vector in f. What is the size of the number of examples l we need to let f' approximate f with a confidence δ and error ϵ. We have that

$P(f) = 1$
$\epsilon = P(f \Delta f')$
(the error is the chance of rejecting an example in f because it is not in f')
$\delta = (1 - \epsilon)^m$
(confidence is the chance of not making an error after learning from l examples)
thus
$\ln\delta \leq l \ln(1 - \epsilon)$
resulting in the following expression for l:

$$l \leq \frac{\ln\delta}{\ln(1 - \epsilon)}$$

This means that the confidence parameter δ and the error parameter ϵ are exponentially small w.r.t. the number of examples l feeded into the learning algorithm. This means that for an arbitrary δ and ϵ we can keep l polynomial because the δ and ϵ decrease exponentially with respect to l.