# On The Learnability Of Disjunctive Normal Form Formulas 

HOWARD AIZENSTEIN<br>aizen@uiuc.edu<br>University of Illinois College of Medicine at Urbana-Champaign<br>190 Medical Sciences Building, 506 S. Mathews, Urbana, Illinois 61801<br>LEONARD PITT<br>pitt@cs.uiuc.edu<br>Department of Computer Science, University of Illinois,<br>1304 W. Springfield Avenue, Urbana, Illinois 61801

Editor: David Haussler


#### Abstract

We present two related results about the learnability of disjunctive normal form (DNF) formulas. First we show that a common approach for learning arbitrary DNF formulas requires exponential time. We then contrast this with a polynomial time algorithm for learning "most" (rather than all) DNF formulas. A natural approach for learning boolean functions involves greedily collecting the prime implicants of the hidden function. In a seminal paper of learning theory, Valiant demonstrated the efficacy of this approach for learning monotone DNF, and suggested this approach for learning DNF. Here we show that no algorithm using such an approach can learn DNF in polynomial time. We show this by constructing a counterexample DNF formula which would force such an algorithm to take exponential time. This counterexample seems to capture much of what makes DNF hard to learn, and thus is useful to consider when evaluating the run-time of a proposed DNF learning algorithm. This hardness result, as well as other hardness results for learning DNF, relies on the construction of particular hard-to-learn formulas, formulas that appear to be relatively rare. This raises the question of whether most DNF formulas are learnable. For certain natural definitions of "most DNF formulas," we answer this question affirmatively.


Keywords: DNF formulas, PAC learning, concept learning, computational learning theory, greedy heuristic

## 1. Introduction

In a seminal paper, Valiant (1984) introduced a computational model for learning (later termed "PAC", or probably approximately correct, learning). This model captures many common intuitions about learning, and is a useful abstraction for developing provably correct and efficient learning algorithms, as well as for arguing against the existence of such algorithms.
Since the model has been introduced it has been shown that, while the most general classes of boolean formulas are not learnable in polynomial time, a wide variety of restricted classes of formulas are (Angluin, 1992). However, there is still a large gap between these positive and negative learnability results, and lying in this gap are some of the most natural classes of formulas. Principal among these is DNF: disjunctive normal form formulas.

A DNF formula is a boolean formula in which the and's and the or's cannot be arbitrarily nested, but instead can be nested only one level deep - it is the disjunct (or) of a group of conjuncts (ands). For instance, the concept "family pet" might be
described using the DNF formula "(small and dog) or (large and docile and dog) or (declawed and cat) or (with-claws and docile and cat)". As argued by Valiant and others (Valiant, 1985; Michalski et al., 1983; Michalski et al., 1986), DNF is a natural class of formulas, one with which people seem comfortable expressing real concepts. Perhaps due to its naturalness, DNF and its variants have gained wide use in computer systems (i.e., production systems and expert systems), and have been used in many heuristic machine learning systems (Michalski et al., 1983; Michalski et al., 1986). Consequently, it is no surprise that the learnability of DNF has been well-studied.
Although the learnability of general DNF formulas remains open, there are polynomial time algorithms for learning various restricted subclasses of DNF. These subclasses typically arise by limiting various parameters associated with the DNF formulas (Angluin, 1987a; Shackelford \& Volper 1988; Angluin \& Slonim, 1994; Hancock, 1991; Aizenstein \& Pitt 1991; Angluin et al., 1992; Blum \& Rudich, 1992; Goldman \& Mathias, 1992; Aizenstein \& Pitt 1992; Berggren, 1993; Bshouty, 1993; Kushilevitz \& Roth, 1993; Pillapakkamnatt \& Raghavan, 1993; Blum et al., 1994), or by assuming particular distributions on the input examples (Pagallo \& Haussler 1989; Verbeurgt 1990; Hancock \& Mansour, 1991; Flammini et al., 1992; Hancock, 1993; Kushilevitz \& Mansour, 1993; Khardon, 1994; Jackson, 1994). Valiant (1984) and Angluin (1988) have described polynomial-time algorithms that use queries to learn monotone DNF - disjunctive normal form formulas that contain no negated variables. The same natural approach is used in both cases - the algorithm learns by greedily collecting the prime implicants of the unknown formula. This approach is also natural for learning arbitrary DNF, and is at the heart of several conjectured DNF learning algorithms. Further, the underlying goals of several existing machine learning systems appear to be motivated by this greedy search for a small number of prime implicants sufficient to describe the data (Michalski et al., 1983; Michalski et al., 1986; Clark \& Niblett, 1988; Clark \& Boswell, 1991).

Because it is simple, and because it exactly embodies this natural approach, the algorithm described by Valiant (1984) is of particular theoretical interest. This algorithm, which we call learn-DNF, uses subset queries (i.e., it asks whether or not hypothetical terms are implicants of the unknown function) to collect prime implicants.
Although no one has shown whether any of the DNF learning algorithms run in time polynomial in the length of the unknown formula (a standard learning parameter), Valiant has considered more generous parameters for which learn-DNF might run in polynomial time. In particular, Valiant suggests that learn-DNF runs in time polynomial in the length of the largest reduced DNF expression equivalent to the unknown formula. This parameter is referred to as the degree of the unknown formula.
In Section 3 we give a counterexample, a target DNF formula that forces any algorithm (even a probabilistic one) using this approach to take time exponential in the length of the target formula, and even in the degree of the unknown formula - thus contradicting the claim made by Valiant. ${ }^{1}$ While our counterexample does not directly show that machine learning systems such as $A Q$, etc., provably require exponential time to PAC-learn DNF, we suspect that arguments similar to those contained here could be employed to that effect.

The counterexample presented here, as well as other recent results (Angluin and Kharitonov, 1991; Aizenstein, Hellerstein, and Pitt, 1992), suggests that arbitrary DNF formulas are not learnable. These results, however, rely on the construction of particular hard-to-learn DNF formulas, formulas that seem to be relatively rare. This raises the question (which we consider in Section 4) of whether most DNF formulas are learnable. For some, seemingly natural, definitions of most DNF formulas the answer to this question is trivially "yes". We show that for other, more robust, definitions the answer is also "yes".
The main contributions of this paper appear in Sections 3 and 4, following Section 2, which contains basic definitions and problem statements.

## 2. Definitions

A literal is either a variable $v$ or its negation $\bar{v}$. For a set of variables $\left\{v_{1}, v_{2}, \ldots v_{n}\right\}$, an assignment $a$ is a function $a:\left\{v_{1}, v_{2}, \ldots v_{n}\right\} \rightarrow\{0,1\}$ and extends to all literals (i.e., $a\left(\bar{v}_{i}\right)=1-a\left(v_{i}\right)$ ). A term is a conjunction (.) of literals. A DNF formula is a disjunction ( + ) of terms. Sometimes, to express relationships between literals, terms, and formulas, we will use standard set notation and treat a term as a set of literals and a DNF formula as a set of terms. Assignment $a$ satisfies term $t$ (also written $t(a)=1$ ) iff $(\forall l \in t) a(l)=1$. Assignment $a$ satisfies DNF formula $f$ (written $f(a)=1$ ) iff $(\exists t \in f) t(a)=1$. A term $t$ is an implicant of a DNF formula $f$ if every satisfying assignment for $t$ is also a satisfying assignment for $f$. An implicant $t$ for a DNF formula $f$ is a prime implicant if no proper subset of $t$ is also an implicant of $f$. A DNF formula $f$ is reduced if every term in $f$ is a prime implicant of $f$, and every term is necessary (for every term $t$ in $f$, the formula $f-\{t\}$ expresses a different function than $f$ ). A reduced DNF expression is also referred to as an irredundant sum of prime implicants.
Let $t_{1}$ and $t_{2}$ be terms of a DNF formula $f$ such that there is exactly one literal $l$ such that $l$ is in $t_{1}$ and $\bar{l}$ is in $t_{2}$. The consensus of $t_{1}$ and $t_{2}$ (denoted by $t_{1} \odot t_{2}$ ) is the term consisting of all literals in $t_{1}$ except $l$, and all literals of $t_{2}$ except $l$. For example, $x y \odot \bar{x} z \bar{w}=y z \bar{w}$. The consensus of two terms is only defined when there is exactly one literal that appears negated in one of the terms, and unnegated in the other. Let $t_{3}$ be the consensus of $t_{1}$ and $t_{2}$. It is easily argued that the term $t_{3}$ is an implicant of $f$ and thus can be added to $f$ without changing the function computed by $f$. If we could form the consensus of $t_{3}$ and another term of $f$, this would give us yet another implicant of $f$ that we could add to $f$. A repetition of this process is called iterated consensus of the terms of $f$. We will use the fact that every prime implicant of a formula $f$ can be obtained by iterated consensus of the terms of $f$ (Muroga, 1990).
Notice that if $f$ is a monotone DNF formula (a DNF formula in which no variable appears negated) then consensus is not possible between any two terms of $f$. Further, if we assume that $f$ is reduced, then $f$ is exactly the sum of the prime implicants of $f$ : the sum of prime implicants is the unique reduced DNF representation for $f$. This is the basis of Valiant's monotone DNF learning algorithm. However, when $f$ is a reduced non-monotone DNF formula, iterated consensus may derive additional prime implicants not in $f$. For example, iterated consensus on the reduced DNF formula
$f=\bar{x} \bar{y}+x \bar{z}+y z$ derives the additional prime implicants $x y, \bar{y} \bar{z}$, and $\bar{x} z$. Furthermore, in the non-monotone case, unlike in the monotone case, there is not a unique reduced DNF representation, and the different reduced DNF formulas need not have the same number of terms: both $\bar{x} \bar{y}+x \bar{z}+y z$ and $x y+\bar{x} \bar{y}+y z+\bar{y} \bar{z}$ are reduced DNF formulas for the same function.

### 2.1. The learning models

The two standard learning models we consider in this paper are the PAC model (see Valiant 1984) and the exact model (see Angluin 1988). In one regard, these two learning models are the same: in each of these models a class of formulas $F$ is defined to be learnable when there is an efficient (i.e., polynomial time) algorithm that can learn every particular formula in $F$. The learning models differ, however, in what it means for an algorithm to learn a particular formula $f$. Below we define, for these two models, what it means to learn a particular formula $f$ (chosen from a known class $F$ ).
The size of a boolean formula, denoted by $\|f\|$, is the length of its representation in some natural encoding. Often, we use a more parsimonious size measure for DNF formulas: if $f$ is a DNF formula, then we define the length of $f$ to be the number of terms in $f$ (i.e., $|f|$ ). Since the learning models we consider allow time polynomial in $n$ (as well as in $\|f\| \mid$ ), it follows that, with respect to polynomial time learning, the size measures $|f|$ and $|\mid f \|$ are equivalent. Throughout this paper we assume the standard random access machine (RAM) model of computation, augmented to allow queries (see Angluin, Hellerstein, and Karpinski, 1993).

### 2.1.1. The exact learning model

The exact learning model (Angluin, 1988) requires that after time polynomial in the number of variables $n$ and in the size of $f$, the learning algorithm $A$ outputs a boolean formula $f^{\prime}$ in $F$ that is equivalent to $f$. That is, $f$ and $f^{\prime}$ are satisfied by the same assignments to the variables. In this model, $A$ learns by asking equivalence queries about the target formula: it asks whether a formula $f^{\prime}$ from the class $F$ is equivalent to $f$. If the two formulas are equivalent, then the equivalence query oracle responds "yes," and $A$ has successfully learned $f$. Otherwise, the oracle responds with a counterexample, an assignment that satisfies $f$ or $f^{\prime}$, but not both.

### 2.1.2. The PAC-learning model

The PAC-learning model (Valiant, 1984) requires that, on input of any parameters $\epsilon, \delta$, and $s$ (such that $0<\epsilon, \delta<1$ and $s \geq\|f\|$ ), the learning algorithm $A$ outputs, with probability at least $1-\delta$, a boolean formula $f^{\prime}$ from the class $F$ that $\epsilon$-approximates $f$ : the probability that an assignment (chosen from an arbitrary unknown distribution $D$ ) satisfies $f$ or $f^{\prime}$ but not both is at most $\epsilon$. Further, $A$ must output $f^{\prime}$ in time polynomial in
the number of variables $n, 1 / \epsilon, 1 / \delta$, and $s$. In this model, rather than asking equivalence queries, $A$ is supplied with a source of assignments randomly chosen from $D$, each one labeled as to whether it satisfies the target formula $f$.

### 2.1.3. Membership queries

For many natural classes of formulas, such as DNF, there are no known positive learnability results in either the exact model or the PAC model. Consequently, it is common to enhance these models in a natural way by also allowing the algorithm to ask about particular assignments. Here, in addition to the randomly generated labeled examples or the equivalence queries, we also allow the algorithm to ask an oracle whether a particular assignment satisfies the target formula. We refer to these queries as membership queries, and we refer to the enhanced models, as the PAC+membership and exact+membership learning models.

### 2.1.4. Relationships between the models

Angluin (1987b) has shown how equivalence queries can be replaced with randomly generated labeled examples, thus transforming an exact learning algorithm into a PAClearning algorithm. So any class of formulas learnable in the exact learning model is also learnable in the PAC model. Further, this reduction also works in the presence of membership queries, so any class of formulas that is learnable in the exact+membership learning model is also learnable in the PAC+membership learning model.

### 2.1.5. Other learning models

A number of variations on the above learning models have also been considered (Haussler et al., 1991). Of note is the variation where the learning algorithm is not restricted to output a hypothesis from the target class (learning a class "in terms of" another (cf. Pitt \& Valiant (1988))). For instance, when learning DNF, the algorithm might be allowed to hypothesize a general boolean formula, or perhaps a neural network, rather than being restricted to hypothesizing DNF formulas. The nonlearnability result we present applies only to a particular approach for learning DNF - we defeat algorithms that work by collecting and hypothesizing sets of prime implicants (i.e., DNF formulas). Our result makes no claims about the difficulty of learning DNF using other approaches, such as those which hypothesize from different classes.

## 3. DNF cannot be learned by greedily collecting prime implicants

Here we show that the approach to learning boolean functions of greedily collecting prime implicants does not work for learning arbitrary DNF formulas. We first illustrate this approach to learning, in Section 3.1, using a well-known, straightforward algorithm
for learning monotone DNF. In Section 3.2 we describe the natural extension of this algorithm, learn-DNF, that ostensibly learns the more general class of arbitrary DNF formulas; this is essentially Valiant's (1984) DNF learning algorithm. The following section (Section 3.3) introduces our counterexample DNF formula $f$. We then use $f$ in Section 3.4 to defeat learn-DNF, as well as to defeat certain simple deterministic variants of it. We extend this argument, in Section 3.5, to defeat a wider class of algorithms. This class includes learn-DNF, as well as some probabilistic variants. In Section 3.6 we discuss some implications of these results.

### 3.1. Learning monotone DNF

As mentioned in Section 2, if $f$ is a reduced monotone DNF formula then the sum of all prime implicants of $f$ is its unique reduced DNF representation. Hence, learning monotone DNF by collecting its prime implicants seems like a natural approach, and in fact is the approach used by Valiant (1984) (see also Angluin, 1988, for an analogous algorithm in the exact+membership learning model). Here we briefly describe Valiant's algorithm (Figure 1) and in the following section we describe a generalization of this algorithm that attempts to learn an arbitrary DNF formula.
Suppose the target formula $f^{*}$ (a reduced monotone DNF formula) is defined over the set of variables $V=\left\{v_{1}, \ldots, v_{n}\right\}$. At every iteration of the main loop, the algorithm receives a randomly generated example $a$. If $a$ is a negative example $\left(f^{*}(a)=0\right)$, then the algorithm ignores it. But if $a$ is positive example, then it iteratively flips the values of the variables in $a$ from 1 to 0 to find a minimal positive assignment $a^{\prime}$ : an assignment such that no variable assigned 1 can be flipped to 0 without changing the value of the function from 1 to 0 .
Let variables ( $a^{\prime}$ ) denote the term consisting of exactly those variables that are assigned 1 by $a^{\prime}$. It is straightforward to show that, since $f^{*}$ is monotone, variables $\left(a^{\prime}\right)$ is a prime implicant of $f^{*}$. At all subsequent iterations in which a positive example $a$ is obtained that is not covered by any prime implicant previously found, the algorithm adds a new prime implicant. Together with the fact that the total number of prime implicants of the reduced monotone DNF formula $f^{*}$ is $\left|f^{*}\right|$, this can be used to show that learn-monotone-DNF can learn the target formula in the PAC+membership learning model.

### 3.2. Learning arbitrary $D N F$

The key idea in the above algorithm is to use membership queries to reduce every positive example to a prime implicant of the target formula. This works because, in the monotone case, variables $(a)$ is an implicant of $f^{*}$ exactly when $a$ satisfies $f^{*}$. Thus we can test whether a term is an implicant by asking membership queries. Moreover, variables $(a)$ is a prime implicant whenever $a$ is a minimal positive example (no " 1 " bit can be changed to a " 0 " and still satisfy $f$ ). So, in the monotone case, by iteratively asking a series of membership queries we can reduce a positive example to a prime implicant.

## learn-monotone-DNF

```
\(h \leftarrow \emptyset\)
\(a \leftarrow\) a randomly generated labeled example
IF label \((a)=1\) THEN
(* If \(a\) is a positive example then reduce \(a\) to a minimal positive assignment. *)
        FOR \(i=1\) TO \(n\)
            IF MEMBERSHIP \(\left(a\right.\) with \(v_{i}\) set to 0\()=\) true THEN
            \(a \leftarrow a\) with \(v_{i}\) set to 0
        \(t \leftarrow\) variables \((a)\)
        \(h \leftarrow h \cup\{t\}\)
GOTO 2
```

Figure 1. Algorithm learn-monotone-DNF

However, in the non-monotone case $a$ might satisfy $f^{*}$ even when variables $(a)$ is not an implicant of $f^{*}$. But what if instead we allow the algorithm to ask directly whether a term $t$ is an implicant of $f$ ? This is a special type of subset query (Angluin, 1988). In a subset query we allow the algorithm to ask an oracle whether a particular formula implies the target formula (i.e., is the set of satisfying assignments of the hypothesis a subset of the set of satisfying assignments of the target?). If it does, then the oracle responds "yes"; otherwise it responds "no". In the general subset query model the algorithm can ask about any formula from the target class, but here we are concerned with only a special class of subset queries, those in which the algorithm can ask only about terms. Using these subset queries instead of membership queries, we can now reduce the positive examples to prime implicants as in the monotone case, giving us a general algorithm, learn-DNF (Figure 2), which attempts to learn arbitrary DNF. This is essentially Valiant's (1984) DNF learning algorithm.
Suppose the target formula $f^{*}$ (a reduced DNF formula) is defined over the set of literals $L=\left\{v_{1}, \ldots, v_{n}, \bar{v}_{1}, \ldots, \bar{v}_{n}\right\}$, and let literals $(a)$ be the set of literals in $L$ assigned 1 by a. As in the monotone case, at every iteration of the main loop the algorithm receives a new example $a$. If $a$ is a negative example, then the algorithm ignores it. If $a$ is a positive example, then the algorithm reduces it to a prime implicant of $f^{*}$. But as mentioned earlier, unlike in the monotone case these prime implicants are not necessarily terms of the target DNF formula. Thus it is unclear how many of these prime implicants the algorithm needs to process before learning the target formula. This makes it difficult to argue that the algorithm learns $f^{*}$ in time polynomial in $\left|f^{*}\right|$.
As mentioned earlier, in the non-monotone case, unlike in the monotone case, there is not a unique reduced DNF representation, and the different reduced DNF formulas need not have the same number of terms. This suggests a different learning parameter, the length of the largest reduced DNF formula that expresses $f^{*}$. Valiant refers to this

## learn-DNF

1. $h \leftarrow \emptyset$
$a \leftarrow$ a randomly generated labeled example
2. IF label $(a)=1$ THEN
(* If $a$ is a positive example then reduce $a$ to a minimal positive assignment. *) $t \leftarrow$ literals $(a)$
FOR $i=1$ TO $n$
IF $\operatorname{subset}\left(t-\left\{v_{i}, \bar{v}_{i}\right\}\right)=$ true THEN
$t \leftarrow t-\left\{v_{i}, \bar{v}_{i}\right\}$
$h \leftarrow h \cup\{t\}$
GOTO 2

Figure 2. Algorithm learn-DNF
parameter as the degree of $f^{*}$ and claims that learn-DNF runs in time polynomial in the degree of $f^{*}$. Notice that the degree of $f^{*}$ is at least as large as the length of the smallest DNF formula that expresses $f^{*}$; thus it is at least as generous a parameter. But it is not known how much larger the degree of $f^{*}$ is than the smallest DNF formula for $f^{*}$. If the degree of $f^{*}$ is polynomial in the length of the smallest DNF formula, then presumably learn-DNF would also learn DNF in time polynomial in the standard parameters.
Here we show that, with respect to learning DNF, the relationship between the degree of $f^{*}$ and the size of the smallest DNF for $f^{*}$ does not matter: the algorithm cannot even learn in time polynomial in the degree of $f^{*}$. We describe a reduced DNF formula $f$, which has only one reduced representation, but which has an exponential number of additional prime implicants that do not appear in any reduced DNF representation for $f$. We then show how any algorithm (even a probabilistic one) that uses the above approach of greedily collecting prime implicants can be forced to collect these additional prime implicants. This forces the algorithm to take time exponential in the length of the target formula, and even exponential in the more generous parameter, the degree of $f$.

### 3.3. The counterexample

We construct a formula with $2 m+1$ variables $V=\left\{x_{1}, \ldots, x_{m}, y_{0}, y_{1}, y_{2}, \ldots, y_{m}\right\}$. The formula $f$ constructed is defined by:

$$
f=x_{1} x_{2} \ldots x_{m} y_{0}+\bar{x}_{1} x_{2} \ldots x_{m} y_{1}+\bar{x}_{2} x_{3} \ldots x_{m} y_{2}+\ldots+\bar{x}_{m} y_{m}
$$

For notational convenience we denote the term in $f$ containing $y_{i}$ as $t_{i}$. Below we prove some lemmas that characterize the set of prime implicants of $f$. These are used
later to argue that some natural DNF learning algorithms can be forced to find an exponential number of useless prime implicants of $f$ : prime implicants that cover only an exponentially small region of the distribution of positive examples.

Lemma 1 Every term in $f$ is a prime implicant of $f$.
Proof: Clearly every term in $f$ is an implicant of $f$. Thus, it suffices to show that every term in $f$ is prime - no proper subset of a term is also an implicant.
To see that $t_{0}=x_{1} x_{2} \ldots x_{m} y_{0}$ is prime, notice that the assignment with all the $y$ 's set to 0 and all the $x$ 's set to 1 is a negative example of the function; thus $t_{0}-\left\{y_{0}\right\}$ is not an implicant of $f$. To see that no other proper subset of $t_{0}$ is an implicant of $f$, notice that the assignment with all the $y$ 's except $y_{0}$ set to 0 and only $m-1$ of the $x$ 's set to 1 is a negative example of the function.
Similarly, to see that for every $i$ between 1 and $m-1$, the term $t_{i}=\bar{x}_{i} x_{i+1} \ldots x_{m} y_{i}$ is prime, observe that

1. the assignment with all the $y$ 's set to 0 and all the $x$ 's except $x_{i}$ set to 1 shows that $t_{i}-\left\{y_{i}\right\}$ is not an implicant of $f$,
2. the assignment with all the $y$ 's except $y_{i}$ set to 0 and all the $x$ 's set to 1 shows that $t_{i}-\left\{\bar{x}_{i}\right\}$ is not an implicant of $f$, and
3. the assignment with all the $y$ 's except $y_{i}$ set to 0 and all the $x$ 's except $x_{i}$ and $x_{j}$ (for some $j$ between $i+1$ and $m$ ) set to 1 shows that $t_{i}-\left\{x_{j}\right\}$ is not an implicant of $f$.

To complete the proof notice that the first two cases above also show that $t_{m}$, the only remaining term of $f$, is prime.

Lemma 2 Any sum of prime implicants logically equivalent to $f$ must contain every term in $f$.

Proof: For every term $t$ in $f$ we will construct an assignment $a$ that satisfies $t$, but that does not satisfy any other prime implicant of $f$. Thus, $t$ must be in every sum of prime implicants that expresses $f$.

For every term $t$ in $f$, consider the assignment $a$ that satisfies $t$ and in which every variable not in $t$ is assigned 0 by $a$. Since for every other term $t^{t}$ in $f-\{t\}$ there is a variable $y$ set to 0 by $a, t$ is the only term in $f$ satisfied by $a$.
To see that no other prime implicant of $f$ is satisfied by $a$, observe that every other prime implicant $\tau$ of $f$ must be produced by iterated consensus on some set of at least two distinct terms $\left\{t_{i_{1}}, t_{i_{2}} \ldots, t_{i_{j}}\right\}$ of $f$. Since no variable $y$ appears negated in $f, \tau$ must contain every $y$ that appears in $t_{i_{1}} \cup t_{i_{2}} \cup \ldots \cup t_{i_{j}}$. Thus, $\tau$ must contain at least two distinct $y$ 's, and thus $\tau$ will not be satisfied by $a$.

Definition. Let $S_{m}$ the set of terms given by $S_{m}=\left\{b_{1} b_{2} \ldots b_{m} y_{0}: b_{i}=x_{i}\right.$ or $\left.y_{i}\right\}$.

Lemma 3 Every term in $S_{m}$ is a prime implicant of $f$.
Proof: Let $t$ be an element of $S_{m}$. First we argue, by induction on the number of $y$ 's in $t \cap\left\{y_{1}, y_{2}, \ldots, y_{m}\right\}$, that $t$ is an implicant of $f$.

Base case $\left(\left|t \cap\left\{y_{1}, \ldots, y_{m}\right\}\right|=0\right)$ : In this case $t=x_{1} \ldots x_{m} y_{0}$ is a term of $f$, and thus by Lemma $1 t$ is a (prime) implicant.

Inductive step $\left(\left|t \cap\left\{y_{1}, \ldots, y_{m}\right\}\right|>0\right)$ : Suppose $t \cap\left\{y_{1}, y_{2}, \ldots, y_{m}\right\}=\left\{y_{i_{1}}, y_{i_{2}}, \ldots y_{i_{j}}\right\}$, $j \geq 1$, and $i_{j}$ is the largest index in the set. Then by the inductive hypothesis the term $t^{\prime}=\left(t-\left\{y_{i_{j}}\right\}\right) \cup\left\{x_{i_{j}}\right\}$ (i.e. $x_{i_{j}}$ replaces $y_{i_{j}}$ ) is an implicant. To complete the inductive step, notice that the consensus $t=t^{\prime} \odot t_{i_{j}}$ is an implicant of $f$ - it is the consensus of two implicants.

To complete the proof, we argue that every implicant $t$ in $S_{m}$ is also prime by showing that the term $t^{\prime}$ formed by removing any single variable $v_{i}$ from $t$ is not an implicant of $f$. First we show that no variable $y_{j}$ in $t$ can be removed. The assignment $a$ with all the $x_{i}$ 's except $x_{j}$ set to 1 , and with all the $y_{i}$ 's except $y_{j}$ set to 1 , satisfies this subset of $t$, but $a$ does not satisfy $f$ : every term $t_{l}$ for $0 \leq l<j$ contains $\bar{x}_{l}$ and thus will not be satisfied, $t_{j}$ contains $y_{j}$ and will not be satisfied, and every term $t_{l}$ for $j<l \leq m$ contains $\bar{x}_{l}$ and will not be satisfied. A similar argument shows that no variable $x_{j}$ in $t$ can be removed.

It follows immediately from Lemma 2 that $f$ is the unique reduced DNF formula expressing its function. Hence the degree of $f$ equals $|f|$, which is $m+1$. Further, since $\left|S_{m}\right|=2^{m}$, it follows by Lemma 3 that there are an exponential number of additional prime implicants not in $f$. Notice that these prime implicants cannot appear in any reduced DNF formula for $f$ (because by Lemma 2 every term of $f$ must be included, rendering any additional prime implicants redundant). The central idea of our counterexample is to force learn-DNF and similar algorithms to find these additional prime implicants before finding the essential prime implicants in $f$. We first need the following lemma.

Lemma 4 Every prime implicant $t$ of $f$ not in $S_{m}$ must contain a literal $\bar{x}_{i}$.
Proof: Let $t$ be a prime implicant of $f$. We prove by induction on the minimum number of consensus operations used to produce $t$ from $f$ that either

1. $t \in S_{m}$, or
2. $t$ contains at least one literal $\bar{x}_{i}$, and for every literal $\bar{x}_{i}$ in $t$ the literal $y_{i}$ is also in $t$.

Base case: Suppose no consensus operations are used to produce $t$. Then $t=t_{0}$ or $t \in\left\{t_{1}, \ldots, t_{m}\right\}$. In either case the inductive claim will be satisfied: if $t=t_{0} \in S_{m}$, then the first case of the inductive claim is satisfied, and if $t=t_{i} \in\left\{t_{1}, \ldots, t_{m}\right\}$, then the second case of the inductive claim is satisfied.
Inductive step: Suppose $i$ consensus operations are used to produce $t$. Then $t$ is produced by the consensus of two implicants $t^{\prime}$ and $t^{\prime \prime}$ such that each of $t^{\prime}$ and $t^{\prime \prime}$ result from fewer
than $i$ consensus operations. Thus, by the inductive hypothesis, either both $t^{\prime}$ and $t^{\prime \prime}$ are in $S_{m}$, both $t^{\prime}$ and $t^{\prime \prime}$ satisfy case 2 of the inductive claim, or one of $t^{\prime}$ and $t^{\prime \prime}$ is in $S_{m}$ and the other satisfies case 2 of the inductive claim.
By assumption of this case, consensus must be possible between $t^{\prime}$ and $t^{\prime \prime}$. However, since the terms in $S_{m}$ contain only positive variables, consensus is not possible between any two terms from $S_{m}$. So $t^{\prime}$ and $t^{\prime \prime}$ cannot both be in $S_{m}$.
In the case where both $t^{\prime}$ and $t^{\prime \prime}$ satisfy case 2 of the inductive claim, some literal $\bar{x}_{i}$ must occur in $t^{\prime}$ and some literal $\bar{x}_{j}$ must occur in $t^{\prime \prime}$. Further, since consensus is possible between $t^{\prime}$ and $t^{\prime \prime}$, there must be a choice of $i$ and $j$ such that $i \neq j$. Since the consensus of two terms contains all but one literal from the union of the two terms, at least one of $\bar{x}_{i}$ or $\bar{x}_{j}$ must appear in the resulting term $t=t^{\prime} \odot t^{\prime \prime}$. Since no $\bar{y}_{i}{ }^{\prime}$ s occur in $f$, all the $y_{i}$ 's in $t^{\prime} \cup t^{\prime \prime}$ must also appear in $t$, so for every literal $\bar{x}_{i}$ in $t$ the literal $y_{i}$ is also in $t$.
To complete the inductive step, we consider the case where one of $t^{\prime}$ and $t^{\prime \prime}$ is in $S_{m}$ and the other satisfies case 2 of the inductive claim. Without loss of generality, assume that $t^{\prime} \in S_{m}$. By assumption of this case, consensus is possible between $t^{\prime}$ and $t^{\prime \prime}$. The case where $y_{i} \in t^{\prime}$ and $\bar{y}_{i} \in t^{\prime \prime}$ is not possible because no terms formed by iterated consensus can include a negated $y$ literal. Therefore, there must be a literal $x_{i}$ in $t^{\prime}$ such that $\bar{x}_{i} \in t^{\prime \prime}$. Since $t^{\prime \prime}$ satisfies condition 2 of the inductive claim, it must be the case that $t=t^{\prime} \odot t^{\prime \prime}=\left(t^{\prime} \cup t^{\prime \prime}\right)-\left\{x_{i}, \bar{x}_{i}\right\}$ contains the literal $y_{i}$. Thus, $t$ contains $\left(t^{\prime}-\left\{x_{i}\right\}\right) \cup\left\{y_{i}\right\}$ (i.e., $y_{i}$ replaces $x_{i}$ ), which is in $S_{m}$. Hence, either $t$ is equal to $\left(t^{\prime}-\left\{x_{i}\right\}\right) \cup\left\{y_{i}\right\}$ and thus is in $S_{m}$, or else $t$ properly contains $\left(t^{\prime}-\left\{x_{i}\right\}\right) \cup\left\{y_{i}\right\}$ (which by Lemma 3 is prime) and thus contradicts the assumption that $t$ is prime.

### 3.4. Simple deterministic algorithms

Here we use the counterexample $f$ to show that learn-DNF and certain simple variants of learn-DNF are incapable of learning DNF in polynomial time. The simple variants we consider are those identical to learn-DNF, but in lines 5-7 rather than testing and eliminating the variables according to their standard ordering (which in this case we assume to be $x_{1}, \ldots, x_{m}, y_{0}, y_{1}, \ldots, y_{m}$ ), they test and eliminate the variables using some other fixed order. In the next section we consider a wider class of algorithms.
First we argue in Theorem 1 that, to achieve error less than $\epsilon$, learn-DNF must process at least $(1-\epsilon)\binom{m}{m / 2}$ examples. Since the length of $f$ is linear in $m$ it follows that the algorithm requires exponential time. In Corollary 1 we extend the argument by analogy to simple deterministic variants of learn-DNF.
Define the distribution of examples $D$ to be uniform over the set $A$ of assignments $a$ such that that $a\left(y_{0}\right)=a\left(x_{1}\right)=a\left(x_{2}\right)=\cdots=a\left(x_{m}\right)=1$ and for exactly half of the other $y_{i}$ 's, $a\left(y_{i}\right)=0$. There are exactly $\binom{m}{m / 2}$ assignments in $A$, and since they all satisfy the term $x_{1} \ldots x_{m} y_{0}$, they are all positive examples of $f$.
Our argument that learn-DNF requires exponential time relies on the fact that for every assignment $a$ in $A$, learn-DNF finds a prime implicant that covers only one assignment
in $A$. This follows immediately from the following lemma about the prime implicants found by learn-DNF.

Lemma 5 Suppose $f$ is the target formula and $a \in A$ is the assignment that learn-DNF receives in line 2. Then in lines 5-7, learn-DNF finds the prime implicant $b_{1} \ldots b_{m} y_{0}$ in $S_{m}$, where $b_{i}=y_{i}$ if and only if $a\left(y_{i}\right)=1$.
Proof: Let $a$ be an assignment in $A$. Let $\hat{t}$ be the term found in lines 5-7 of learn-DNF (when $a$ is received in line 2). Let $Y_{1}=\left\{y_{i}: 0 \leq i \leq m \wedge a\left(y_{i}\right)=1\right\}$ and $X_{0}=\left\{x_{i}\right.$ : $\left.1 \leq i \leq m \wedge a\left(y_{i}\right)=0\right\}$. The lemma states that the algorithm finds the prime implicant $X_{0} \cup Y_{1}$ (notice that $y_{0} \in Y_{1}$ ).
In line $4, t$ is initialized as an implicant of $f$. The variables are then sequentially tested (in lines 5-7) and are removed from $t$ only when the resulting term still implies the formula. Thus, the final term, $\hat{t}$, will be an implicant of $f$. Moreover, no unnecessary variable will remain - so, $\hat{t}$ will be a prime implicant.

We complete the proof of this lemma by showing that $\hat{t}$ must contain $X_{0}$ and $Y_{1}$, and thus must be the prime implicant $X_{0} \cup Y_{1}$. Notice that $\hat{t} \subseteq$ literals $(a)$. So, $a$ must satisfy $\hat{t}$. Since $a \in A, a\left(x_{1}\right)=\cdots=a\left(x_{m}\right)=1$. Thus, by Lemma 4 , only prime implicants in $S_{m}$ are satisfied by $a$. It is straightforward to verify that of the prime implicants in $S_{m}, a$ satisfies only those of the form $b_{1} \ldots b_{m} y_{0}$ in $S_{m}$, where $b_{i}=x_{i}$ if $a\left(y_{i}\right)=0$. Consequently, $\hat{t}$ must contain $X_{0}$.
Since the $x$ variables are always tested before the $y$ variables (in line 6), and since $X_{0} \cup Y_{1}$ is an implicant of $f$, the algorithm will necessarily eliminate all of the $x$ variables that are not in $X_{0}$. After testing the $x$ variables in line 6 , the resulting term $t$ is implied only by prime implicants that contain all the literals in $Y_{1}$, So $\hat{t}$ must contain $Y_{1}$.

THEOREM 1 learn-DNF must process $(1-\epsilon) 2^{\left(\begin{array}{c}m / 2\end{array}\right)}$ examples of $f$ from $A$ to achieve error less than $\epsilon$ on $f$ with respect to $D$.
Proof: Recall that $A$ contains $\binom{m}{m / 2}$ assignments, all of which are positive examples. Thus to achieve error less than $\epsilon$ with respect to $D$, learn-DNF must return a formula $h$ that covers at least $(1-\epsilon)\binom{m}{m / 2}$ assignments in $A$. By Lemma 5 , the prime implicant added to $h$ due to example $a$ is satisfied by only one assignment in $A$ : the assignment $a$. Thus to cover $(1-\epsilon)\binom{m}{m / 2}$ assignments in $A$, learn-DNF must process at least that many.

Corollary 1 Suppose Simple-Variant is a learning algorithm identical to the algorithm learn-DNF, except in lines $4-6$ rather than testing and eliminating the variables in the order $v_{1}, \ldots, v_{n}$ (the standard order), Simple-Variant tests and eliminates the variables in some other fixed order, $g$. Then there exists a function $f^{\prime}$ and a distribution $D^{\prime}$ such that Simple-Variant must process $(1-\epsilon)\binom{m}{m / 2}$ examples of $f^{\prime}$ from $D^{\prime}$ to achieve error less than $\epsilon$ on $f^{\prime}$ with respect to $D^{\prime}$.

Proof: We define the function $f^{\prime}$ and the distribution of assignments $D^{\prime}$ to be the same as $f$ and $D$ except with variables $\left(x_{1}, \ldots, x_{m}, y_{0}, \ldots, y_{m}\right)$ replaced respectively with
the same variables, but reordered according to $g$. Notice that the behavior of learn-DNF learning $f^{\prime}$ using the ordering $g$ is the same as the behavior of learn-DNF learning $f$ using the original ordering. The corollary now follows from Theorem 1 .
The distribution $D$ is surprising in that it generates only positive examples and hence a minor modification of learn-DNF could learn this distribution by simply hypothesizing "true". However, a simple modification of $D$ could defeat the new algorithm. Yet, there are numerous other ways that learn-DNF might be modified to learn the particular distribution $D$. For example, the order of testing variables in line 5 can be modified so that Lemma 5 no longer holds. In this case alternative distributions (variants of $D$ ) can be constructed to defeat each of the new algorithms. In fact, in the following section we show that distributions exist to defeat even a probabilistic ordering of the variables. The arguments we give, however, do not intend to show that no algorithm can learn DNF in polynomial time; rather we argue only that no algorithm using the same general approach of learn-DNF (i.e., greedily collecting the prime implicants) can learn DNF in polynomial time.

### 3.5. A wider class of algorithms

Our argument to defeat the previous algorithms took advantage of the fact that those algorithms eliminated the variables in a fixed order. Typically randomized algorithms are used to defeat worst-case scenarios such as this. The question arises whether a variant of learn-DNF that finds prime implicants by eliminating the variables in a random order could avoid the worst-case problem of finding useless prime implicants, and perhaps even learn the general class of arbitrary DNF formulas.
Here we show that this is not the case. We consider a wider class of algorithms where the order is not fixed, but is any probabilistic function of the current example. In particular, we model a probabilistic choice of ordering by a deterministic function $g$ that takes as input a positive example $a$ and a random string $r$ (which has length at most $p(n)$ for some polynomial $p$ ), and outputs a permutation $g(a, r)$ of the variables. For any such $g$, the algorithm learn- $\mathrm{DNF}_{g}$ (shown in Figure 3) is the same as the algorithm learn-DNF, but replaces the ordering $v_{1}, \ldots, v_{n}$ with a random ordering $g(a, r)$.
The argument to defeat such a randomized algorithm is similar to the deterministic case: now rather than constructing a counterexample $f$ to defeat a fixed ordering, we define a distribution of functions $D_{F}$ and argue that with high probability a function randomly chosen from $D_{F}$ will defeat the algorithm, and now rather than arguing that every prime implicant found is satisfied by only one example in $A$, we argue that with high probability every prime implicant found is satisfied by only a small fraction of the relevant set of assignments.

### 3.5.1. Definitions

Suppose $V$ is the set of $n$ variables $\left\{v_{1}, \ldots, v_{n}\right\}$. An ordering function $g$ (with respect to $V$ ) is a function that maps every assignment $a$ and random string $r$ (a string from

## learn- $\mathrm{DNF}_{g}$

$h \leftarrow \emptyset$
2. $a \leftarrow$ a randomly generated labeled example
3. IF label $(a)=1$ THEN
(* If $a$ is a positive example then reduce $a$ to a minimal positive assignment. *)
4. $\quad t \leftarrow$ literals $(a)$
5. CHOOSE a random bit string $r$ of length $p(n)$
6. Sequentially, for every $v$ in $g(a, r)$
7. IF $\operatorname{subsex}(t-\{v, \bar{v}\})=$ true THEN
8. $\quad t \leftarrow t-\{v, \bar{v}\}$
9. $h \leftarrow h \cup\{t\}$
10. GOTO 2

Figure 3. Algorithm learn- $\mathrm{DNF}_{g}$
$\{0,1\}^{p(n)}$ for some polynomial $p$ ) to an ordering $\sigma$ of the $n$ variables in $V$. Let $\mathcal{L}=$ \{learn- $\mathrm{DNF}_{g}: g$ is an ordering function and learn- $\mathrm{DNF}_{g}$ is the algorithm shown in Figure 3.\}.

A variable-permutation $\pi$ of the set of variables $V=\left\{x_{1}, \ldots, x_{m}, y_{0}, \ldots, y_{m}\right\}$ is a one-to-one mapping of the variables $\left\{x_{1}, \ldots, x_{m}, y_{0}, \ldots, y_{m}\right\}$ onto itself. If $t$ is a term, then $\pi(t)$ denotes the term obtained by replacing every variable $v$ in $t$ with the variable $\pi(v)$. If $f$ is a set of terms, then $\pi(f)$ denotes the set of terms obtained by replacing every term in $f$ with the term $\pi(t)$. A variable-permutation $\pi$ of $V$ is a "swapping" variable-permutation if for every $i$ between 1 and $m, \pi\left(x_{i}\right)=x_{i}$ or $y_{i}$, and $\pi\left(y_{i}\right)=y_{i}$ or $x_{i}$. We define the set of formulas $F=\{\pi(f): \pi$ is a swapping variable-permutation of $V\}$ and the distribution $D_{F}$ of formulas as uniform over the set of formulas in $F$.

We also define the distribution of assignments $D_{f_{\pi}}$ as uniform on the set $A_{\pi}$ of assignments $a_{\pi}$ such that $a\left(\pi\left(y_{0}\right)\right)=a\left(\pi\left(x_{1}\right)\right)=a\left(\pi\left(x_{2}\right)\right)=\ldots a\left(\pi\left(x_{m}\right)\right)=1$, and for exactly half of the other $y_{i}$ 's, $a\left(\pi\left(y_{i}\right)\right)=0$. There are $\binom{m}{m / 2}$ such assignments, and since they all satisfy the term $\pi\left(x_{1} \ldots x_{m} y_{0}\right)$, they are all positive examples of $\pi(f)$.

### 3.5.2. The argument

Consider the case where the target formula $\pi(f)$ is randomly chosen from the distribution $D_{F}$, and the assignments are randomly chosen from the distribution $D_{\pi(f)}$. We show that every learning algorithm learn- $\mathrm{DNF}_{g}$ in $\mathcal{L}$ is incapable of PAC-learning $D N F$ in polynomial time by arguing that for any polynomial $\varphi(|V|,|f|, 1 / \epsilon, 1 / \delta)$, and any $\delta$ and $\epsilon$, with probability at least $\delta$, learn- $\mathrm{DNF}_{g}$ requires more than $\varphi(|V|,|f|, 1 / \epsilon, 1 / \delta)$ examples to achieve error less than $\epsilon$ on $\pi(f)$.

First we establish in Lemma 6 that the probability that learn- $\mathrm{DNF}_{g}$ reduces an example $a$ to a useful prime implicant (a prime implicant which covers more than an exponentially small fraction of the assignments in $A_{\pi}$ ) is very small. From this we prove (Corollary 2) that after processing a polynomial number of assignments there is only a very small probability that learn- $\mathrm{DNF}_{g}$ will have found a useful prime implicant. This is used to prove Theorem 2.

LEMMA 6 Suppose $g$ is an ordering function, the target formula $\pi(f)$ is randomly chosen from the distribution $D_{F}$, and in line 2 of learn- $D N F_{g}$ the assignment $a$ is randomly chosen from the distribution $D_{\pi(f)}$. Then the probability that lines 6-8 of learn-DNF $g_{g}$ find a prime implicant that contains fewer than $m / 8$ variables $z$ such that $z \in\left\{\pi\left(y_{1}\right), \pi\left(y_{2}\right), \ldots, \pi\left(y_{m}\right)\right\}$ is at most $e^{-m / 32}$.
Proof: Let $I=\left\{i: a\left(y_{i}\right)=a\left(x_{i}\right)=1\right\}$ (i.e. the set of indices $i$ such that $x_{i}$ and $y_{i}$ are both set to 1 ). Using an argument analogous to that used to prove Lemma 5 , it is straightforward to show that lines $6-8$ of learn- $\mathrm{DNF}_{g}$ find a prime implicant $\tau=$ $y_{0} b_{1} \ldots b_{m}$ in $\left\{\pi(t): t \in S_{m}\right\}$, where $b_{i}=y_{i}$ if and only if $i \in I$ and $x_{i}$ appears before $y_{i}$ in $g(a, r)$. Thus, the probability that $\tau$ contains fewer than $m / 8$ variables $z$ such that $z \in\left\{\pi\left(y_{1}\right), \pi\left(y_{2}\right), \ldots, \pi\left(y_{m}\right)\right\}$ is the probability that for fewer than $m / 8$ indices $i$ in $I$, $\pi\left(x_{i}\right)$ appears before $\pi\left(y_{i}\right)$ in $g(a, r)$.
Consider some integer $i$ in $I$. We will show that the probability that $\pi\left(x_{i}\right)$ appears before $\pi\left(y_{i}\right)$ in $g(a, r)$ is $1 / 2$ by considering the distribution of possible variablepermutations. It is clear that for a swapping variable-permutation $\pi, x_{i}$ occurring before $y_{i}$ corresponds to $\pi\left(x_{i}\right)$ appearing before $\pi\left(y_{i}\right)$, exactly when $\pi\left(x_{i}\right)=x_{i}$. Further, it is also clear that there are the same number of swapping variable permutations $\pi$ such that $\pi\left(x_{i}\right)=x_{i}$ as there are swapping variable permutations $\pi^{\prime}$ such that $\pi^{\prime}\left(x_{i}\right)=y_{i}$. Thus, since the swapping variable-permutations are randomly chosen from the uniform distribution, there is an equal chance that a particular order of elimination $g(a, r)$ corresponds to a $\pi\left(x_{i}\right)$ being removed before a $\pi\left(y_{i}\right)$ as vice-versa. Similarly, if $j \in I-\{i\}$, then the event that $\pi\left(x_{j}\right)$ is eliminated before $\pi\left(y_{j}\right)$ also has probability $1 / 2$ and is independent from the event that $\pi\left(x_{i}\right)$ is eliminated before $\pi\left(y_{i}\right)$.
To complete the proof of this lemma, we bound the probability $q$ that for fewer than $m / 8$ indices $i$ in $I, \pi\left(x_{i}\right)$ is eliminated before $\pi\left(y_{i}\right)$. This is the probability that for $m / 2$ independent trials, each with probability of success equal to $1 / 2$, there are fewer than $m / 8$ successes. It follows, using Chernoff bounds (see Angluin \& Valiant, 1979, Proposition 2.4(a)), that $q \leq e^{-\beta^{2} m / 8}$ where $\beta$ is the fraction that the outcome deviates from the expected. Since the expected number of successes is $m / 4, \beta=(m / 8) /(m / 4)$. Thus, $q \leq e^{-(1 / 4)(m / 8)}=e^{-m / 32}$.

COROLLARY 2 Suppose $g$ is an ordering function, the target formula $\pi(f)$ is randomly chosen from the distribution $D_{F}, \varphi(|V|,|f|, 1 / \epsilon, 1 / \delta)$ is a polynomial, $M$ is a set of $\varphi(|V|,|f|, 1 / \epsilon, 1 / \delta)$ assignments randomly chosen from $D_{\pi(f)}$, and the learning algorithm learn-DNF $F_{g} \in \mathcal{L}$. Then the probability that some assignment $a \in M$ will result in a prime implicant $\tau$ being found in lines $6-8$ of the algorithm such that $\tau$ contains fewer than $m / 8$ variables $z$ in $\left\{\pi\left(y_{1}\right), \pi\left(y_{2}\right), \ldots, \pi\left(y_{m}\right)\right\}$ is at most $\varphi(|V|,|f|, 1 / \epsilon, 1 / \delta) e^{-m / 32}$.

Proof: By Lemma 6 the probability that any particular assignment in $M$ results in a prime implicant being found that satisfies the constraints of this lemma is at most $e^{-m / 32}$. Since each assignment in $M$ is chosen independently from $D_{\pi(f)}$, the probability of any one resulting in a prime implicant satisfying the conditions of this lemma is at most $\varphi(|V|,|f|, 1 / \epsilon, 1 / \delta) e^{-m / 32}$.

Theorem 2 There does not exist an algorithm learn-DNFg in $\mathcal{L}$ that can PAC-learn DNF in time polynomial in the length of the target formula, or even in the degree of the target formula.
Proof: Suppose contrary to this theorem that some algorithm learn- DNF $_{g}$ in $\mathcal{L}$ learns DNF in time $\varphi(|V|,|f|, 1 / \epsilon, 1 / \delta)$, and let $\pi(f)$ be a formula randomly chosen from the distribution $D_{F}$. By Corollary 2 , with probability at least $1-\varphi(|V|,|f|, 1 / \epsilon, 1 / \delta) e^{-m / 32}$ every prime implicant found by learn- $\mathrm{DNF}_{g}$ has at least $m / 8$ variables $z$ such that $z \in\left\{\pi\left(y_{1}\right), \pi\left(y_{2}\right), \ldots, \pi\left(y_{m}\right)\right\}$. Since $|V|=2 m+1$ and $|f|=m+1$, this probability can be rewritten as $1-\varphi(2 m+1, m+1,1 / \epsilon, 1 / \delta) e^{-m / 32}$. Clearly for any $\delta<1$ there exists an $m^{\prime}$ such that for any $m \geq m^{\prime}, 1-\varphi(2 m+1, m+1,1 / \epsilon, 1 / \delta) e^{-m / 32}>\delta$. Thus for sufficiently large $m$, with probability greater than $\delta$, every prime implicant found by the algorithm has at least $m / 8$ variables $z$ such that $z \in\left\{\pi\left(y_{1}\right), \pi\left(y_{2}\right), \ldots, \pi\left(y_{m}\right)\right\}$. The number of assignments of $A_{\pi}$ that satisfy such a prime implicant is at most

$$
\binom{m-m / 8}{m / 2-m / 8} \leq\binom{ 7 m / 8}{3 m / 8} \leq(7 / 8)^{m / 2}\binom{m}{m / 2}=(7 / 8)^{m / 2}\left|A_{\pi}\right| .
$$

As in the proof of Theorem 1, to achieve error at most $\epsilon$ with respect to $D$, learn- DNF $_{g}$ must return a formula $h$ that covers at least $(1-\epsilon)\left|A_{\pi}\right|$ assignments in $D$. Thus, with probability greater than $\delta$, the algorithm requires at least $(1-\epsilon)\left|A_{\pi}\right| /\left((7 / 8)^{m / 2}\left|A_{\pi}\right|\right)=$ $(1-\epsilon)(8 / 7)^{(m / 2)}$ examples to have error at most $\epsilon$. Thus learn- $-\mathrm{DNF}_{g}$ requires time exponential in $|f|=m+1$. Furthermore, by Lemma 2, the degree of $f$ is equal to $|f|$, so learn- $\mathrm{DNF}_{g}$ also requires time exponential in the degree of $f$. Finally note that not only does learn- $\mathrm{DNF}_{g}$ fail to PAC-learn some particular function $f$, but with high probability it fails to learn a randomly chosen one.

### 3.6. Discussion

We have shown that in the PAC model the natural approach to learn DNF by collecting prime implicants is bound to fail. Similar constructions show that this approach is also bound to fail in the exact learning model (introduced by Angluin, 1988). Our counterexamples, however, do not show that DNF is not learnable. Note that the class $F$ of hard to learn counterexamples is easily learnable by an algorithm that first looks at a collection of examples to determine an appropriate ordering function, and then reduces the examples using this ordering. This suggests that learning DNF requires a more global approach, such as an Occam algorithm (see Blumer et al., 1987), which first collects a large sample of examples and then tries to find a small formula consistent with them. The general polynomial time learning of DNF, thus, remains a central open problem.

It also still remains open whether DNF is learnable in time polynomial in its degree, and it remains open whether the degree of a DNF formula is polynomial in its smallest DNF representation - perhaps this is true only for DNF formulas whose smallest representation is polynomial in the number of variables $n$. These learning problems are open even if we allow the algorithm to ask subset or membership queries (in addition to randomly generated labeled examples).
Initially we observed a slightly simpler version of this counterexample ( $x_{1} \cdots x_{m}+$ $\bar{x}_{1} y_{1}+\cdots+\bar{x}_{m} y_{m}$ ), which has many of the same properties as the formula discussed here, but which defeats only those algorithms in $\mathcal{L}$ having a constant ordering function. This has proved to be a useful counterexample to various algorithms suggested by colleagues. Our counterexamples seem to capture much of what makes DNF hard to learn. They suggest that learning DNF, according to the standard definitions, may be even harder than previously thought. In the following section we consider some alternative definitions for which DNF is learnable.

## 4. Learning most DNF formulas

According to the standard PAC and exact learning models, for an algorithm to learn a class of formulas, such as DNF, the algorithm must be capable of learning any particular formula from the class. Thus, as was done in the previous section, one can show that the class of DNF formulas is difficult to learn by describing a particular DNF formula, or a set of DNF formulas, that is difficult to learn. Similarly, previous results supporting the conjecture that DNF is not learnable, also relied on the construction of particular hard-to-learn formulas. These hard-to-learn formulas seem to be somewhat contrived and not representative of most DNF formulas. In this section we consider modifying the standard exact learning model and we ask about learning most, rather than all, DNF formulas.
What does it mean to learn most DNF formulas? One natural definition is to consider the set $F_{n}$ of all well-formed DNF expressions over some set of $n$ variables, and ask whether all but an exponentially small fraction of the formulas in $F_{n}$ are learnable. According to this definition, almost all of the formulas in $F_{n}$ have an exponential (in $n$ ) number of terms. Since the learning algorithm is allowed time polynomial in the size of the target formula, it will now be allowed time exponential in $n$. This makes learning essentially trivial: a simple, brute force equivalence query learning algorithm (discussed in Section 4.2) can learn DNF in time polynomial in $2^{n}$.
The problem with the above definition is that it allows the algorithm to ignore the DNF formulas we are most interested in learning, DNF formulas whose number of terms is a polynomial in $n$. One way to avoid this problem is to parameterize on the number of terms, and ask about learning most DNF formulas with a bounded number of terms. As before, by most formulas we mean all but an exponentially small fraction: most formulas in a distribution of formulas $F_{n}$ have a property if the probability that a formula $f$ randomly chosen from $F_{n}$ does not have the property is at most $1 / 2^{\Omega(n)}$. Here we consider this alternate model, and we answer the question affirmatively. In

Section 4.4 we present a membership and equivalence query algorithm for learning most DNF formulas within this model, and we prove:

Theorem 3 There exists a polynomial $\varphi(n, s)$ such that if $n$ and $s$ are positive integers, and $X=\left\{x_{1}, \ldots, x_{n}\right\}$ is a set of variables, then for most DNF formulas with at most $s$ terms, learn-most-DNF (Figure 6) finds an equivalent DNF formula in $\varphi(n, s)$ time.

This model, however, also has a limitation. Our proof of Theorem 3 relies on the observation that essentially all terms have $\Omega(n)$ variables. Using this observation we show that most formulas have a special property (they are distance-2), and are thus exactly learnable in polynomial time. In our model most terms have $\Omega(n)$ variables since we randomly construct each term by flipping a coin to decide for every variable $v$ whether it is in that term, and then flipping a coin to decide whether to include the literal $v$ or the literal $\bar{v}$.
The observation that most terms have $\Omega(n)$ variables can also be used to show that for any fixed distribution, with high probability (over choices of DNF formulas, made after the distribution is chosen), the hypothesis "false" will be accurate - since with high probability, most terms in the DNF formulas will have size $\Omega(n)$ and thus each DNF will be satisfied by only few positive examples.
These observations suggest a more general definition for most DNF formulas that parameterizes on the probability $p$ that a particular variable is in the term. Thus, whereas our model considers learning only formulas where $p$ is set to $1 / 2$, this new model considers learning for arbitrary values of $p$; it asks whether for arbitrary values of $p$ (and $s$ ) most DNF formulas are learnable. This seems to more completely capture the question about learning most DNF formulas; unlike the other models it does not take undue advantage of the fact that most formulas are large or that most terms are long. We do not, however, have a complete answer to this question. Instead we point out (in Section 4.5) some values of $p$ for which most DNF formulas are learnable.
The remainder of this section is organized as follows. First, we introduce some definitions and prove two key properties about certain DNF formulas (Section 4.1). Next (Section 4.2) we present an algorithm, learn-large-DNF, for learning DNF formulas that have an exponential number of terms. The algorithm learn-large-DNF exactly learns all DNF formulas $f$ in $O\left(2^{2 n} n\right)$ time (Theorem 4). Thus it learns, in polynomial time, DNF formulas with an exponential (in $n$ ) number of terms. A different learning algorithm, learn-distance-2-DNF, is presented in Section 4.3. This algorithm, suggested by Angluin (1990), learns in polynomial time a special case of disjoint DNF (distance-2 DNF). Since most not-so-large DNF formulas are distance-2 (Theorem 5), it follows that most not-solarge DNF formulas are polynomial time learnable. The algorithms learn-large-DNF and learn-distance-2-DNF are combined to form a general DNF learning algorithm, learn-most-DNF, which is presented in Section 4.4. We then use the results of Sections 4 and 4.3 to prove Theorem 3. In Section 4.5 we conclude by discussing some limitations and implications of this result.

### 4.1. Some definitions and properties

Denote assignment $a$ with literal $x$ fixed to 0 as $a_{x \leftarrow 0}$. Throughout, $a_{x \leftarrow 0}$ will represent $a$ with the truth value of $x$ flipped, since we will write " $a_{x \leftarrow 0}$ " only when $a$ assigns $x$ to 1 . The sensitive set (of literals for $a$ ) is defined by sensitive $(a)=\{x: a(x)=1$ and $\left.f(a) \neq f\left(a_{x \leftarrow 0}\right)\right\}$, thus flipping the value of any $x$ in sensitive $(a)$ causes $f$ to change values. Observe that the sensitive set of an assignment can be found by asking $n+1$ membership queries: With one membership query we find the value of $f(a)$ and then with $n$ additional membership queries we ask for each literal $x$ assigned 1 by a (i.e., literals $(a)$ ), whether changing the value assigned to $x$ causes the value of the function to change.
Let $X=\left\{x_{1}, \ldots, x_{n}\right\}$ denote a set of variables. Define $\mathcal{T}_{n}$ as the set of terms over $X$. For every subset of the variables $X^{\prime} \subseteq X, \mathcal{T}_{n}$ contains each of the $2^{\left|X^{\prime}\right|}$ terms composed of the variables in $X^{\prime}$ (e.g., $x_{1} x_{2}, x_{1} \bar{x}_{2}, \bar{x}_{1} x_{2}$, and $\bar{x}_{1} \bar{x}_{2}$ are the four terms composed of variables in $X^{\prime}=\left\{x_{1}, x_{2}\right\}$ ). As mentioned in Section 2, we sometimes treat a term $t$ as a set of literals; thus $|t|$ denotes the number of literals in $t$.
For every positive integer $n$, we define the distribution of terms $D\left(\mathcal{T}_{n}\right)$. A term $t$ is randomly chosen from $D\left(\mathcal{T}_{n}\right)$ by randomly choosing (with probability $1 / 2$ ), for every variable $v_{i}$, whether $v_{i}$ is in $t$. If $v_{i}$ is chosen to be in $t$, then whether the variable $v$ appears as the literal $v$ or as the literal $\bar{v}$ is also decided randomly, with probability $1 / 2$. Thus, if $t$ is randomly chosen from $D\left(\mathcal{T}_{n}\right)$, then the probability that some literal $l$ appears in $t$ is $1 / 4$. This seems to be a fairly natural distribution of terms; it is the one we will consider throughout most of this section. However, as pointed out previously, a preponderance of the terms in $\mathcal{T}_{n}$ have $\Omega(n)$ literals; thus this distribution allows us to essentially ignore terms with fewer literals. In Section 4.5 we consider other distributions that do not have this limitation: we consider a different distribution for every probability $p$ that a variable occurs in a term.
For every positive integer $s$, we form the distribution of formulas formulas ${ }_{n, s}$. A formula $f$ is chosen from formulas ${ }_{n, s}$ by randomly choosing (with replacement) $s$ terms from $D\left(\mathcal{T}_{n}\right)$.
Let $t$ and $t^{\prime}$ be terms. We define distance $\left(t, t^{\prime}\right)$ to be the number of distinct variables $x_{1}, \ldots, x_{k}$ such that for each $i, x_{i}$ appears in one of $t$ and $t^{\prime}$, and $\bar{x}_{i}$ appears in the other. A DNF formula $f$ is distance- $k$ if for every pair of distinct terms $t$ and $t^{\prime}$ in $f$, distance $\left(t, t^{\prime}\right) \geq k$. Notice that the class of distance-1 DNF formulas is the same as the class of disjoint DNF formulas.

Proposition 1 If $f$ is any DNF formula and $a$ is an assignment such that $f(a)=1$, then sensitive $(a) \subseteq \cap\{t: t(a)=1\}$.

Proof: Suppose to the contrary that, for some literal $x$ in sensitive(a), there exists a term $t$ satisfied by $a$ such that $x$ is not in $t$. Then $t$ would also be satisfied by $a_{x \leftarrow 0}$, which implies that $f\left(a_{x \leftarrow 0}\right)=1$, and which thus contradicts the assumption that $x \in$ sensitive $(a)$.

## learn-large-DNF

1. $H \leftarrow \emptyset$
2. LOOP until Equivalence-Query $(H)$ returns true Let $a$ be the (positive) counterexample returned by Equivalence-Query $(H)$
3. $\quad t \leftarrow \operatorname{literals}(a)$
4. $H \leftarrow H \cup\{t\}$
5. CONTINUE

Figure 4. Algorithm learn-large-DNF

Proposition 2 Suppose $f$ is a distance- 2 DNF formula over the set of variables $X$ and $a$ is a positive example of $f$. Then a satisfies exactly one term $t$ of $f$, and sensitive $(a)=t$.

Proof: Since $f$ is a distance-2 DNF formula no two terms can be satisfied by the same assignment. Thus, the assignment $a$ satisfies exactly one term $t$ of $f$. Furthermore, by Proposition $1, \operatorname{sensitive}(a) \subseteq t$. To see that $\operatorname{sensitive~}(a)=t$, consider why some literal $x$ in $t$ might not be in sensitive( $a$ ). This can happen only if when we flip the value assigned to $x$ by $a$ the value of the function does not become 0 (i.e., $f\left(a_{x \leftarrow 0}\right)=f(a)=1$ ). Clearly $t$ is no longer satisfied when we flip $x$ (since $x \in t$ ), so some other term $t^{\prime} \neq t$ must be satisfied by $a_{x \leftarrow-0}$. But then distance $\left(t, t^{\prime}\right) \leq 1$, which contradicts the assumption that $f$ is a distance-2 DNF formula.

### 4.2. Very large DNF formulas are learnable

In Figure 4, we present an algorithm learn-large-DNF, which (as argued in Theorem 4) learns DNF formulas in $2^{2 n} n$ time. This algorithm is later used as a routine by our algorithm learn-most-DNF; it is used for learning, in polynomial time, those DNF formulas that have more than $2^{n / 64}$ terms.

Theorem 4 If $f$ is a DNF formula over the set of variables $X=\left\{x_{1}, \ldots, x_{n}\right\}$, then learn-large-DNF will find an equivalent DNF formula in $O\left(2^{2 n} n\right)$ time.

Proof: In each iteration the algorithm simply memorizes a new positive example (of which there are at most $2^{n}$ ) as an individual term. Hence there are at most $2^{n}$ iterations. Each iteration takes time $O(n)$ plus the time to make an equivalence query, which may vary from $O(n)$ to $O\left(2^{n} n\right)$, depending on the computational model.

## learn-distance-2-DNF

1. $H \leftarrow \emptyset$
2. LOOP until Equivalence-Query $(H)$ returns true Let $a$ be the (positive) counterexample returned by Equivalence-Query $(H)$ $H \leftarrow H \cup\{$ sensitive $(a)\}$.
3. $\quad$ 4. CONTINUE

Figure 5. Algorithm learn-distance-2-DNF

### 4.3. Most of the remaining DNF formulas are learnable

In Section 4.2 we presented an algorithm, learn-large-DNF, which learns DNF formulas in time polynomial in $2^{n}$. Thus, for DNF formulas that have an exponential (in $n$ ) number of terms, learn-large-DNF runs in time polynomial in the size of the target formula (one of the standard learning parameters). Here we show that most of the remaining DNF formulas can also be learned in polynomial time. We first show that distance-2 DNF formulas are learnable in polynomial time (Theorem 5), and then we argue that most of the remaining DNF formulas are distance-2 (Theorem 6 and Corollary 3).

THEOREM 5 If $f$ is a distance- 2 DNF formula over the set of variables $X$, then learn-distance-2-DNF (see Figure 5) will find an equivalent DNF formula in time $O\left(|f|^{2} n\right)$.

Proof: As with algorithm learn-large-DNF, learn-distance-2-DNF continues running until an equivalent DNF formula is hypothesized in line 2, so if it halts, then it is correct. We bound the number of iterations of the main loop. Notice that the only terms ever in $H$ are those added in line 4 (when $a$ is a positive example). By Proposition 2 these must be terms from $f$. Thus $H$ always contains a subset of the terms in $f$. So in every iteration of the main loop the counterexample $a$ must be a positive counterexample the assignment $a$ satisfies a term $t$ that is in $f$, but that is not in $H$.
Furthermore, by Proposition 2 the term sensitive (a), which is added to $H$ in line 3, is exactly the single term of $f$ satisfied by $a$. Thus after at most $|f|$ iterations of the main loop, $H$ will contain every term of $f$, and the subsequent equivalence query will return true. The theorem follows from the observation that the sensitive set operation requires time $O(n)$, and thus every iteration of the main loop requires time $O(n|f|)$.

To prove Theorem 6 (that most DNF formulas with a bounded number of terms are distance-2) we need the following claims. In Claim 1 we use Chernoff bounds to establish that a preponderance of the terms in $\mathcal{T}_{n}$ have at least $n / 4$ literals. Claim 1 is used in the proof of Claim 2 to bound the probability that two terms, randomly (and independently) chosen from $D\left(\mathcal{T}_{n}\right)$ have distance less than 2 . The proof of Theorem 6 then follows.

Claim 1 Let $n$ be a positive integer, and be a term randomly chosen from $D\left(\mathcal{T}_{n}\right)$. Then the probability that $|t|<n / 4$ is at most $e^{-n / 16}$.

Proof: Suppose the term $t$ is randomly chosen from the distribution $D\left(\mathcal{T}_{n}\right)$. We bound the probability $q$ that $t$ contains fewer than $n / 4$ literals. This is the probability that for $n$ independent trials (one for each of the $n$ variables), each with probability of success equal to $1 / 2$, there are fewer than $n / 4$ successes. It follows, using Chernoff bounds (Angluin \& Valiant, 1979, Proposition 2.4(a)), that $q \leq e^{-\beta^{2} n / 4}$ where $\beta$ is the fraction that the outcome deviates from the expected. In this case, $\beta=(n / 4) /(n / 2)$. So $q \leq e^{-(1 / 4)(n / 4)}=e^{-n / 16}$.

Claim 2 Suppose $n$ is a positive integer, and $t$ and $t^{\prime}$ are two terms randomly (and independently) chosen from $D\left(\mathcal{T}_{n}\right)$. Then the probability that distance $\left(t, t^{\prime}\right)<2$ is at $\operatorname{most} 2\left(e^{-n / 16}\right)+n(3 / 4)^{n / 4-1}$.

Proof: First consider the case where both $|t|$ and $\left|t^{\prime}\right|$ are at least $n / 4$. Since $t$ and $t^{\prime}$ were chosen independently, for every variable $x$ the conditional probability $\operatorname{Pr}\left(x \in t^{\prime} \mid x \in t\right)$ is at least $1 / 2$ (it is not exactly $1 / 2$ since we are considering only terms with more than $n / 4$ literals). Likewise, for every literal $x$ the conditional probability $\operatorname{Pr}\left(\bar{x} \in t^{\prime} \mid x \in t\right)$ is at least $1 / 4$. Thus, the probability that for some set $S$ of $|t|-1$ literals in $t$ no literal in $S$ appears negated in $t^{\prime}$ is at most $(3 / 4)^{|t|-1}$. Hence, the probability that there exists a set of $|t|-1$ such literals in $t$ is at most $|t|(3 / 4)^{|t|-1} \leq n(3 / 4)^{n / 4-1}$.
To complete the proof of this claim, we show that the probability that the above case does not occur is at most $2 e^{-n / 16}$. Notice that the only remaining case is when at least one of $t$ and $t^{\prime}$ has length less than $n / 4$. It follows from Claim 1 that the probability that this happens is at most $2 e^{-n / 16}$.
TheOrem 6 Suppose $n$ and $s$ are positive integers, and the formula $f$ is randomly chosen from formulas ${ }_{n, s}$. Then the probability that $f$ is not distance-2 is at most $s^{2}\left[2\left(e^{-n / 16}\right)+n(3 / 4)^{n / 4-1}\right]$.

Proof: Since each term in $f$ is chosen randomly (with replacement) from $D\left(\mathcal{T}_{n}\right)$, we can apply Claim 2 to bound the probability that any of the at most $s^{2}$ pairs of terms in $f$ fail to be distance-2. Hence, the probability that $f$ is not distance- 2 is at most $s^{2}\left[2\left(e^{-n / 16}\right)+n(3 / 4)^{n / 4-1}\right]$.
Corollary 3 Suppose $n \geq 64$ and $s \leq 2^{n / 64}$ are both positive integers, and the formula $f$ is randomly chosen from formulas $s_{n, s^{.}}$. Then the probability that $f$ is not distance- 2 is at most $n 2^{-(n / 32-2)}$.

Proof: The probability that $f$ is not distance- 2 satisfies the following bounds:

$$
\begin{array}{rlrl}
\operatorname{Pr}[f \text { is not distance-2 }] & \leq s^{2}\left[2 e^{-n / 16}+n(3 / 4)^{n / 4-1}\right. & & \text { (by Theorem 6) } \\
& \leq 2^{n / 32} 2\left(e^{-n / 16}\right)+2^{n / 32} n(3 / 4)^{n / 4-1} & \left(\text { since } s \leq 2^{n / 64}\right) \\
& \leq 2^{-(n / 32-1)}+2^{n / 32} n(3 / 4)^{n / 4-1} & \\
& \leq 2^{-(n / 32-1)}+2^{-n / 32} n & \\
& \leq n 2^{-(n / 32-2)} &
\end{array}
$$

### 4.4. The algorithm and proof of its correctness

Figure 6 gives the algorithm learn-most-DNF for learning most DNF formulas. To prove Theorem 3, we exhibit a polynomial $\varphi(n, s)$ such that for every $s$, most of the formulas $f$ in formulas ${ }_{n, s}$ will be found by learn-most-DNF in $\varphi(n, s)$ time. First consider the case where $s<2^{n / 64}$. By Corollary 3 most of the formulas in formulas ${ }_{n, s}$ are distance-2; thus by Theorem 5 these are learnable in time $|f|^{2} n$. Since (for sufficiently large $n$ ) $2^{2 n}>\left(2^{n / 64}\right)^{2} n>|f|^{2} n$, it follows that most of the formulas in formulas $s_{n, s}$ will be learned in line 1 of learn-most-DNF in polynomial time.
Next we consider the case where $s \geq 2^{n / 64}$. It is straightforward to show that most formulas in $f_{n, s}$ contain at least $2^{n / 128}$ variable occurrences. From Theorem 4 it follows that if the algorithm has not learned the target formula in line 1 of learn-most-DNF, then it must learn the formula in line 2. To complete the proof of the second case, and thus the theorem, notice that the run-time of learn-most-DNF is bounded by the run-time of line 1 + the run-time of line $2=O\left(|f|^{2} n\right)+O\left(2^{2 n} n\right)$, which is polynomial in $|f| \geq 2^{n / 128}$.

## learn-most-DNF

1. Run learn-distance-2-DNF for at most $2^{2 n}$ iterations or until the equivalence-query returns true.
2. If an equivalent DNF formula was not found in line 1 , then run learn-large-DNF until an equivalent DNF formula is found.

Figure 6. Algorithm learn-most-DNF

### 4.5. Limitations and implications of this result

As mentioned earlier, this result is limited in that the distribution of formulas we consider, formulas $_{n, s}$, contains a preponderance of formulas whose terms all have size $\Omega(n)$; formulas that contain terms of size $o(n)$ are essentially ignored. One way around this limitation is to consider instead, distributions that focus on a different average term size, and then ask whether, with respect to each of those distributions, most DNF formulas are learnable.
We construct these different distributions by parameterizing based on the probability $p$ that a particular variable occurs in a term. Thus, rather than resulting in an average term size of $n / 2$ (as was the case with $D\left(\mathcal{T}_{n}\right)$ ), we now have an average term size of $n p$. For every probability $p$, we form the distribution of terms terms $s_{n, p}$. A term $t$ is chosen from terms $_{n, p}$ by independently choosing, with probability $p$, for every variable $x$ in $X$, whether $x \in t$. As before, if the variable $x$ is chosen to be in $t$, then we randomly choose (with probability $1 / 2$ ) whether $x$ or $\bar{x} \in t$. For every positive integer $s$, we form
the distribution formulas ${ }_{n, s, p}$ of formulas as before, but now we choose from terms $s_{n, p}$, rather than from $D\left(\mathcal{T}_{n}\right)$.
It remains open whether a general theorem, analogous to Theorem 3, holds for this new model. To prove such a theorem one would need to show that there exists a polynomial time algorithm that learns most of the DNF formulas in each of the distributions formulas $_{n, s, p}$. In Theorem 3 we only showed that this holds when we parameterize on $n$ and $s$ but leave $p$ fixed at $1 / 2$. It is straightforward to generalize our theorem for any constant probability $p$ (in this case the algorithm is allowed time exponential in $1 / p$ ). Thus when the average term size is a constant fraction of $n$ (i.e., $n p$ ) most DNF formulas are learnable.
Similarly, if we allow $p$ to be any constant ratio of $n$ (i.e., if for some constant $c$, $p=c / n$ ), then polynomial time learning is also possible. In fact, in this case we can exactly learn in polynomial time with equivalence queries alone, without using membership queries. This is possible because when $p=c / n$ the average term size is a constant, $n(c / n)=c$, and a polynomial time algorithm has been given, by Valiant (1984), for learning the class of $k$-DNF (DNF formulas in which the term size is bounded by the constant $k$ ).
Thus learning most DNF formulas is possible when the average term size is at most a constant, or at least a constant fraction of $n$. It remains open, however, whether most DNF formulas are learnable when the average term size is asymptotically larger than a constant, yet smaller than a constant fraction of $n$ (i.e., when $p$ is chosen such that $\omega(1) \leq n p \leq o(n))$.

## Acknowledgements

This work was supported in part by NSF Grant IRI-9014840, and was completed while the first author was in the Department of Computer Science, University of Illinois at Urbana-Champaign.

## Notes

1. If instead we slightly modify the definition of degree to be the length of the largest reduced DNF expression that implies the target formula (an alternative definition suggested by Valiant, 1993), then it remains open whether Valiant's conjecture holds. However, it is straightforward to show that the conjecture holds if degree is defined to be the length of the sum of all the prime implicants implied by the unknown formula.

## References

Aizenstein, H., Hellerstein, L., \& Pitt, L. (1992). Read-thrice DNF is hard to learn with membership and equivalence queries. In Proceedings of the 33 rd Annual IEEE Symposium on Foundations of Computer Science, Washington, D.C., October 1992. IEEE Computer Society Press.
Aizenstein, H. \& Pitt, L. (1991). Exact learning of read-twice DNF formulas. In Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, Washington, D.C., October 1991. IEEE Computer Society Press.

Aizenstein, H. \& Pitt, L. (1992). Exact learning of read- $k$ disjoint DNF and not-so-disjoint DNF. In Proceedings of the Sth Annu. Workshop on Comput. Learning Theory, New York, NY, August 1992. ACM Press.
Angluin, D. (1987a). Learning k-term DNF Formulas using Queries and Counterexamples. Tech. rep., Department of Computer Science, Yale University, Report No. RR-559, 1987.
Angluin, D. (1987b). Learning regular sets from queries and counterexamples. Information and Computation, 75(2): 87-106.
Angluin, D. (1988). Queries and concept learning. Machine Learning, 2:319-342.
Angluin, D. (1992). Computational learning theory: Survey and selected bibliography. In Proceedings of the 24th Annual ACM Symposium on Theory of Computing, pages 351-369, New York, May 1992. ACM press.
Angluin, D., Frazier, M., \& Pitt, L. (1992) Learning conjunctions of Horn clauses. Machine Learning, 9:147-164, 1992.
Angluin, D., Hellerstein, L., \& Karpinski, M. (1993). Learning read-once formulas with queries. J. ACM, 40:185-210, 1993.
Angluin, D., \& Kharitonov, M. (1991). When won't membership queries help? In Proceedings of the 23 rd Annual ACM Symposium on Theory of Computing, New York, May 1991. ACM Press.
Angluin, D., \& Pitt, L. (1990). Private communication.
Angluin, D. \& Slonim, D. K. (1994). Randomly fallible teachers: Learning monotone DNF with an incomplete membership oracle, Machine Learning, 14:7-26. Previously appeared in Proc. 4th Annu. Workshop on Comput. Learning Theory.
Angluin, D. \& Valiant, L. G. (1979). Fast probabilistic algorithms for hamiltonian circuits and matchings. J. Comp. Sys. Sci., 18(2):155-193.
Berggren, U. (1993). Linear time deterministic learning of $k$-term DNF In Proceedings of the 6th Annu. Workshop on Comput, Learning Theory, New York, NY, August 1993. ACM Press.
Blum, A., \& Rudich, S. (1992). Fast learning of $k$-term DNF formulas with queries. In Proceedings of the Twenty-Fourth ACM Symposium on Theory of Computing, pages 382-389. ACM Press.
Blum, A., Khardon, R., Kushilevitz, E., Pitt, L., \& Roth, D. (1994). On Learning Read- $k$-Satisfy- $j$ DNF. In Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory, pages 110-117, New Brunswick, New Jersy, July, 1994. ACM Press
Blumer, A., Ehrenfeucht, A., Haussler, D. \& Warmuth, M. (1987). Occam's razor. Information Processing Letters, 24:377-380.
Bshouty, N. H. (1993) Exact learning via the monotone theory. In Proceedings of the IEEE Symp. on Foundations of Computer Science, pages 302-311, Palo Alto, CA., 1993. IEEE Computer Society Press
Clark, P. \& Boswell R., (1991). Rule induction with CN2. In Proceedings of the European Working Session on Learning, Berlin, 1991. Springer-Verlag.
Clark, P. \& Niblett, T. (1989). The CN2 Induction Algorithm. Machine Learning, 3:261-283.
Flammini, M., Marchetti-Spaccamela, A. \& Cera, L. (1992). Learning DNF formulae under classes of probability distributions. In Proceedings of the 5th Annu. Workshop on Comput. Learning Theory, New York, NY, August 1992. ACM Press.
Goldman, S. \& Mathias, H. (1992). Learning $k$-term DNF formulas with an incomplete membership oracle. In Proceedings of the Sth Annu. Workshop on Comput. Learning Theory, New York, NY, August 1992. ACM Press.
Hancock, T. (1991). Learning $2 \mu$-DNF formulas and $k \mu$ decision trees. In Proceedings of the 4th Annu. Workshop on Comput. Learning Theory, San Mateo, CA, August 1991. Morgan Kaufmann.
Hancock, T. (1993). Learning $k \mu$ decision trees on the uniform distribution. In Proceedings of the 6th Annu. Workshop on Comput. Learning Theory, New York, NY, August 1993. ACM Press.
Hancock, T. \& Mansour, Y. (1991). Learning monotone $k \mu$-DNF formulas on product distributions. In Proceedings of the 4 th Annu. Workshop on Comput. Learning Theory, San Mateo, CA, August 1991. Morgan Kaufmann.
Jackson, J. (1994). An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. To appear, Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, Santa Fe, New Mexico, November 1994. IEEE Computer Society Press.
Khardon, R. (1994). On using the Fourier transform to learn disjoint DNF. Information Processing Letters, 49(5):219-222, March 1994.
Kushilevitz, E., \& Mansour, Y. (1993). Learning decision trees using the Fourier spectrum. Siam Journal of Computing, 22(6):1331-1348, 1993. Earlier version appeared in Proc. 23rd Ann. IEEE Symp. on Foundations of Computer Science, 1991.

Kushilevitz, E. \& Roth, D. (1993). On learning visual concepts and DNF formulae. In Proceedings of the Gth Annu. Workshop on Comput. Learning Theory, New York, NY, August 1993. ACM Press. To appear, Machine Learning.
Michalski, R.S., Carbonell, J.G. \& Mitchell, T.M. (1983). Machine Learning: An Artificial Intelligence Approach (Volume 1). Tioga Publishing Co., Pala Alto, California.
Michalski, R.S., Carbonell, J.G. \& Mitchell, T.M. (1986). Machine Learning: An Artificial Intelligence Approach (Volume 2). Morgan Kaufmann, Los Altos, California.
Muroga, S. (1990). Logic Design and Switching Theory. Robert E. Krieger Publishing Co., Malabar Florida. Pagallo, G. \& Haussler, D. (1989). A greedy method for learning $\mu$-DNF functions under the $u$ niforn distribution. Tech. rep., University of California at Santa Cruz, Report No. UCSC-CRL-89-12, 1989.
Pillapakkamnatt, K. \& Raghavan, V. (1993). Read twice DNF formulas are properly learnable. Technical Report TR-CS-93-59, Vanderbilt University, Computer Science Department, 1993. To appear, Proceedings of the 1st European Conference on Computational Learning Theory (EuroColt 93).
Shackelford, G. \& Volper, D. (1988). Learning $k$-DNF with Noise in the Attributes. In Proceedings of the lst Annu. Workshop on Comput. Learning Theory, San Mateo, CA, August 1988. Morgan Kaufmann.
Valiant, L.G. (1984). A theory of the learnable. Communications of the ACM, 27(11):1134-1142.
Valiant, L.G. (1985). Learning disjunctions of conjunctions. In Proceedings of the 9th $n$ International Joint Conference on Artificial Intelligence, Vol. I, pages 560-566, Los Angeles, California.
Valiant, L.G. (1993). Private communication.
Verbeurgt, K. (1990). Learning DNF under the uniform distribution in quasi-polynom ial time. In Proceedings of the 3rd Annu. Workshop on Comput. Learning Theory, San Mateo, CA, August 1990. Morgan Kaufmann.

Received September 29, 1993
Accepted August 11, 1994
Final Manuscript October 25, 1994

