On Learning Read-k-Satisfy-j DNF

Howard Aizenstein[*] Avrim Blum[†] Roni Khardon[‡]
Eyal Kushilevitz[§] Leonard Pitt[¶] Dan Roth[||]

December 20, 1995

Abstract

We study the learnability of Read-k-Satisfy-j (RkSj) DNF formulas. These are boolean formulas in disjunctive normal form (DNF), in which the maximum number of occurrences of a variable is bounded by k, and the number of terms satisfied by any assignment is at most j. After motivating the investigation of this class of DNF formulas, we present an algorithm that for any unknown RkSj DNF formula to be learned, with high probability finds a logically equivalent DNF formula using the well-studied protocol of equivalence and membership queries. The algorithm runs in polynomial time for $k \cdot j = O(\frac{\log n}{\log \log n})$, where n is the number of input variables.

Keywords: DNF, learning, computational learning theory, decision trees.

[*]School of Medicine University of Pittsburgh Western Psychiatric Institute and Clinic 3811 O'Hara St. Pittsburgh, PA 15213 E-mail: aizen+@pitt.edu Research supported in part by NSF Grant IRI-9014840.

[†]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3891. E-mail: avrim@cs.cmu.edu. Research supported in part by NSF National Young Investigator grant CCR-93-57793 and a Sloan Foundation Research Fellowship.

[‡]Aiken Computation Lab., Harvard University, Cambridge, MA 02138. E-mail: roni@das.harvard.edu. Research supported by grant DAAL03-92-G-0115 (Center for Intelligent Control Systems).

[§]Department of Computer Science, Technion, Haifa, Israel 32000. E-mail: eyalk@cs.technion.ac.il. http://www.cs.technion.ac.il/~eyalk. Part of this research was done while the author was at Aiken Computation Laboratory, Harvard University, supported by research contracts ONR-N0001491-J-1981 and NSF-CCR-90-07677.

[¶]Department of Computer Science, University of Illinois, Urbana, IL 61801. E-mail: pitt@cs.uiuc.edu. Research supported in part by NSF grant IRI-9014840.

[||]Dept. of Appl. Math. & CS, Weizmann Institute of Science, Rehovot 76100, ISRAEL. E-mail: danr@wisdom.weizmann.ac.il. This work was done while at Harvard University, supported by NSF grant CCR-92-00884 and by DARPA AFOSR-F4962-92-J-0466.

1 Introduction

A central question in the theory of learning is to decide which subclasses of boolean formulas can be learned in polynomial-time with respect to any of a number of reasonable learning protocols. Among natural classes of formulas, those efficiently representable in disjunctive normal form (DNF) have been extensively studied since the seminal paper of Valiant [Val84]. Nonetheless, whether for such formulas there are learning algorithms that can be guaranteed to succeed in polynomial time using any of the standard learning protocols remains a challenging open question.

Consequently, recent work has focused on the learnability of various restricted subclasses of DNF formulas. For example, the learnability of those DNF formulas with a bounded number of terms, or with bounded size terms, or with each variable appearing only a bounded number of times, has received considerable attention (see e.g. [Val84, Ang87, PV88, BS90, Han91, Aiz93, PR95]). Investigation along these lines is of interest for two reasons: First, techniques developed for learning subclasses of DNF may well be applicable to solving the more general problem, just as difficulties encountered may suggest approaches towards proving that the general problem admits no tractable solution. Second, and at least as important from a practical perspective, is that in many real-world machine-learning settings the full generality of DNF may not be required, and the function to be learned might have an efficient expression in one of the restricted forms for which learning algorithms have been found.

In this paper we show (Theorem 19) that boolean formulas over n variables that can be expressed as *read-k satisfy-j DNF* (which we abbreviate "RkSj") are efficiently learnable, provided that $k \cdot j = O(\frac{\log n}{\log\log n})$. An RkSj DNF representation for a function is a DNF formula in which the maximum number of occurrences of each variable is bounded by k and each assignment satisfies at most j terms of the formula. The special case of RkSj for $j = 1$ is referred to as *disjoint* DNF, and we denote it as RkD. The motivation for considering disjoint DNF expressions arises from the fact that they generalize the decision-tree representation of boolean functions, which have been used extensively in machine-learning theory and practice [BFOS84, Qui86, Qui93]. The "Satisfy-j" constraint relaxes the notion of disjointness, thus incorporating a wider class of functions. In the next section we will discuss the relationships between RkSj DNF, RkD DNF, and previously investigated subclasses of DNF formulas.

The learning model we use is the standard equivalence and membership queries model [Ang88]: We assume some unknown *target* RkSj formula F to be learned is chosen by nature. A learning algorithm may propose as a hypothesis any DNF formula H by making an *equivalence query* to an oracle. (This has sometimes been referred to as an "extended" equivalence query, as the hypothesis of the algorithm need not be in read-k satisfy-j form.) If H is logically equivalent to F then the answer to the query is "yes" and the learning algorithm has succeeded and halts. Otherwise, the answer to the equivalence query is "no" and the algorithm receives a *counterexample* – a truth assignment to the variables that satisfies F but does not satisfy H (a *positive* counterexample) or vice-versa (a *negative* counterexample). The learning algorithm may also query an oracle for the value of the function F on a particular assignment (example) a by making a *membership query* on a. The response to such a query is "yes" if a is a positive example of F ($F(a) = 1$), or "no" if a is a negative example of F ($F(a) = 0$).

We say that the learner *learns* a class of functions $\mathcal{F}$, if for every function $F \in \mathcal{F}$ and for any confidence parameter $\delta > 0$, with probability at least $1 - \delta$ over the random choices of the learner, the learner outputs a hypothesis H that is logically equivalent to F and does so in time polynomial in n (the number of boolean variables of F), $|F|$ (the length of the shortest representation of F in some natural encoding scheme), and $1/\delta$. This protocol is a probabilistic generalization of the slightly more demanding exact learning model, which requires that deterministically, in time polynomial in n and $|F|$, the learning algorithm necessarily finds some logically equivalent H. Learnability in this model implies learnability in the well-studied "PAC" model with membership queries [Val84, Ang88]. In that model, instead of equivalence queries, examples from an arbitrary, unknown probability distribution $\mathcal{D}$ are available, and the goal of the learner is to find with probability at least $1 - \delta$ a hypothesis H that disagrees with F on a set of assignments of probability at most ϵ measured according to $\mathcal{D}$.

The rest of the paper is organized as follows: We review related work in Section 2. Section 3 includes preliminary definitions. In Section 4 we prove some useful properties of RkSj DNF formulas. In Section 5 we present the learning algorithm and prove its correctness. In Section 6 we show that R2D DNF does not have small CNF representation (and in particular no small decision trees), which demonstrates that the class of R2D DNF (and the more general class RkSj) is incomparable to subclasses of DNF recently shown learnable.

2 Related results

When membership queries are not available to the learning algorithm, the problem of learning DNF, or even restricted subclasses of DNF, appears quite difficult. In the model of learning with equivalence queries only, or in the PAC model without membership queries [Val84], learning algorithms are known only for k-DNF formulas (DNF formulas with a constant number of literals in each term) [Val84, Ang88], DNF formulas with a constant number of terms [KLPV87, PV88] (provided that the hypotheses need not be in the same form), or for "polynomially explainable" sub-classes of DNF in which the number of possible terms causing an example to be positive is bounded by a polynomial [Val85, KR93]. Using information theoretic arguments, Angluin [Ang90] has shown that DNF in general is not learnable in polynomial-time using just DNF equivalence queries, although the negative result does not extend to the PAC model.

In contrast, there are a number of positive learning results when membership queries are allowed, and in many of these cases it can be shown that without the membership queries the learning problem is as hard as the problem of learning general DNF. Among these learnable subclasses are: Monotone DNF [Val84, Ang88], Read twice DNF [Han91, AP91, PR95], "Horn" DNF (at most one literal negated in every term) [AFP92], $\log n$ term DNF [BR92], and DNF $\cap$ CNF (this class includes decision trees) [Bsh95].

Read-k-DNF Particularly relevant to our work is the considerable attention that has been given to the learnability of boolean formulas where each variable occurs some bounded (often constant) number k of times ("read k"). Recently, polynomial-time algorithms have been given for learning read-once and read-twice DNF [AHK93, Han91, AP91, PR95]. (The result of [AHK93] is actually much stronger, giving an algorithm for learning any read-once boolean formula, not just those efficiently representable as read-once DNF.)

The learnability of read-k DNF for $k \geq 3$ seems to be much more difficult. Recent results show that in the PAC model with membership queries learning read-thrice DNF is no easier than learning general DNF formulas [AK95, Han91]. In the PAC model, assuming the existence of one-way functions, Angluin and Kharitonov [AK95] have shown that membership queries cannot help in learning DNF (assuming that the learning algorithm is not required to express its hypotheses in any particular form, and that the distribution on examples is "polynomially bounded"). In other words, assuming one-way functions exist, in the PAC model the problem of learning read-thrice DNF with membership queries is no easier than learning general DNF without queries (in polynomially bounded distributions). In the model of learning from equivalence and membership queries, it has been shown ([PR94], see also [AHP92]) that there is no polynomial-time algorithm for exactly learning read-thrice DNF using read-thrice DNF equivalence queries, unless P = NP. One way to view our results is to note that while the work referenced above indicates that for $k \geq 3$ learning read-k DNF is difficult, by adding a disjointness condition on the terms of the formula (or even a much weaker condition that each assignment satisfies only j terms) the learning problem becomes tractable.

Disjoint DNF Variations of disjoint DNF formulas (without any "read" restriction) have also been considered, as disjoint DNF formulas are a natural generalization of decision trees. While every boolean function clearly has a disjoint DNF expression (simply take a fundamental product of exactly n literals for each satisfying assignment), a function with a short DNF representation might have an exponentially long disjoint representation. However, every decision tree can be efficiently expressed as a disjoint DNF formula, by creating a term corresponding to the assignment of variables along each branch that leads to a leaf labeled "1". Recently, Bshouty [Bsh95] has given an algorithm (using equivalence and membership queries) for learning boolean formulas in time polynomial in the size of their DNF and CNF representations. Since every decision tree has a DNF and CNF that have size polynomial in the size of the original tree, his algorithm may be used to learn decision trees. We show (Theorem 20) that even read-2 disjoint DNF formulas do not necessarily have small CNF representations (and therefore do not have small decision trees). In particular, we present a family of functions $\{F_n\}$ that have short ($poly(n)$) read-2 disjoint DNF formulas but require CNF formulas of size $2^{\Omega(\sqrt{n})}$.

Other results concerning the learnability of disjoint DNF have also been obtained. On the positive side, Jackson [Jac94] gives a polynomial-time algorithm for learning arbitrary DNF formulas in the PAC model with membership queries, provided that the error is measured with respect to the uniform distribution. This extends results on learning decision trees, disjoint DNF and Satisfy-j DNF formulas in the same model [KM93, Kha94, BFJ$^+$94]. On the negative side, some recent hardness results for learning DNF in restricted models apply for disjoint DNF as well: Blum et al. [BFJ$^+$94] prove a hardness result for learning $\log n$ disjoint DNF in the "statistical queries" model (note that in this model the learner does not have access to a membership oracle). Aizenstein and Pitt [AP95] show that even if subset queries are allowed, disjoint DNF (and also read-twice DNF) is not learnable by algorithms that collect prime implicants in a greedy manner. Yet, the learnability for disjoint DNF remains unresolved in any reasonable learning model, other than the result above assuming a uniform distribution. Another way, then, to view our results, is to note that if we add a read-k restriction on disjoint DNF, we obtain a positive learning result.

Finally, note that RkSj DNF may be thought of as a generalization of k-term DNF (those DNFs with at most k terms): Every k-term DNF formula is trivially an RkSk-DNF formula. Thus, our results may also be viewed as an extension of previous results for learning k-term DNF formulas [Ang87, BR92, Bsh95], although a true generalization of the strongest of these results would learn RkSj DNF for $k + j = O(\log n)$. We leave the latter as an interesting open problem.

3 Preliminaries

Let $x_1, \ldots, x_n$ be n boolean variables. A *literal* is either a variable x_i or its negation $\overline{x_i}$. A literal ℓ is said to have *index* i if $\ell \in \{x_i, \overline{x_i}\}$. A *term* is a conjunction (AND) of literals; a DNF formula is a disjunction (OR) of terms. For convenience, we will also treat a set of literals as the term that consists of the conjunction of literals in the set, and will treat a set of terms as the DNF formula obtained by taking the disjunction of terms in the set. An *example* (or *assignment*) a is a boolean vector over $\{0,1\}^n$, and we write $a[i]$ to denote the ith bit of a. An example a satisfies a term t (denoted by $t(a) = 1$) if and only if it satisfies all the literals in t. An example a satisfies a DNF formula F (written $F(a) = 1$) if and only if there is at least one term $t \in F$ that a satisfies. If a satisfies F then a is said to be a *positive example* of F, otherwise it is a *negative example*. An RkSj DNF formula F is a DNF formula in which each variable appears at most k times, and for every assignment a, there are at most j terms of F satisfied by a.

For two boolean functions F_1, F_2 we write that F_1 implies F_2 to mean that for every assignment a, if $F_1(a) = 1$ then $F_2(a) = 1$. A term t is an *implicant* of a DNF formula F if t implies F; it is a *prime implicant* if no proper subset of t is also an implicant of F.

We often want to look at examples obtained by changing the value of a given example on a specified literal. Toward this end, we introduce the following notation: Let $a \in \{0,1\}^n$ be an example, and i an index. We define the example $b = \mathit{flip}(a, i)$ by: $b[i] = 1 - a[i]$ and $b[i'] = a[i']$ for $i' \neq i$. For instance, if $a = 1101$ then $\mathit{flip}(a, 3) = 1111$. If ℓ is a literal of index i, then $\mathit{flip}(a, \ell)$ is defined to be the example $\mathit{flip}(a, i)$. Hence, if $a = 1101$, then $\mathit{flip}(a, x_3) = 1111$ and also $\mathit{flip}(a, \overline{x_3}) = 1111$. Usually, we will write $\mathit{flip}(a, \ell)$ only when assignment a satisfies literal ℓ. Finally, we may apply the flip operator to sets of literals as well, to obtain an assignment given by simultaneously flipping each literal in the set: For any set $I \subseteq \{1, \ldots, n\}$ define $\mathit{flip}(a, I)$ to be the assignment b such that $b[i] = 1 - a[i]$ if $i \in I$, and $b[i'] = a[i']$ for $i' \notin I$.

4 Properties of RkSj DNF formulas

A natural way to gather information about a DNF formula F by asking membership queries is to find assignments a such that when the assignment to some single literal of a is changed, the value of F changes. That is, for some literal ℓ, $F(\mathit{flip}(a, \ell)) \neq F(a)$. Such an assignment is called a *sensitive* assignment, and all such literals ℓ are sensitive literals for a. More formally:

Definition 1 *For an assignment $a \in \{0,1\}^n$ and a function F, the set of* sensitive *literals is given by $sensitive(a) = \{\ell \mid F(\mathit{flip}(a,\ell)) \neq F(a)\}$.*

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 determine whether the value of $F(\mathit{flip}(a,\ell))$ differs from $F(a)$ for each of the n literals satisfied by a.

Property 2 *If F is any DNF formula and a is an assignment such that $F(a) = 1$, then $sensitive(a) \subseteq \cap\{t \in F : t(a) = 1\}$. In other words, each literal in $sensitive(a)$ appears in all terms satisfied by a.*

Proof: Suppose to the contrary that for some literal ℓ in $sensitive(a)$ and for some term t satisfied by a, ℓ is not in t. Then t would also be satisfied by $\mathit{flip}(a,\ell)$, which implies that $F(\mathit{flip}(a,\ell)) = 1$, contradicting the assumption that $\ell \in sensitive(a)$. ■

Property 2 suggests that one way to find (at least part of) a term of a DNF formula, given a satisfying assignment a, is to construct the sensitive set of a. In the event that the sensitive set is *exactly* a term t satisfied by a, then we have made significant progress, and can continue to find other terms. However, Property 2 only guarantees that each sensitive literal is in every term a satisfies; it does not guarantee that *every* literal of *some* term is in the sensitive set. Below, we will show that for RkSj DNF formulas, we can find assignments whose sensitive sets are not missing "many" literals. From these assignments, we will be able to find terms of the formula.

Suppose a satisfies only a single term t. By Property 2, any literal ℓ that is not in t will not be in $sensitive(a)$. But why might some literal ℓ that *is* in t fail to be in $sensitive(a)$? It must be because $\mathit{flip}(a,\ell)$ satisfies some term t' containing $\overline{\ell}$. Thus, t' was "almost" satisfied by assignment a; only the literal ℓ was set wrong. More formally:

Definition 3 *A term t is* almost satisfied *by an assignment a with respect to index i if $t(a) = 0$ but $t(\mathit{flip}(a,i)) = 1$. A term t is* almost satisfied *by an assignment a with respect to (literal) ℓ if a satisfies literal ℓ, and if $t(a) = 0$ but $t(\mathit{flip}(a,\ell)) = 1$.*

Let $Y(a)$ be the set of all i's such that some term t is almost satisfied by a with respect to i. The following lemma is central in the analysis of our algorithm; it gives a bound on the size of $Y(a)$. In the appendix it is shown that this bound is essentially optimal.

Lemma 4 *Let F be an RkSj formula, and let $a \in \{0,1\}^n$. Then, $|Y(a)| \leq 2kj$.*

Proof: Let $a \in \{0,1\}^n$ be fixed. Note that for each almost satisfied term t, there is exactly one index i such that t is almost satisfied (by a) with respect to i. We partition the set of almost satisfied terms into $m = |Y(a)|$ nonempty equivalence classes $S_{i_1}, S_{i_2}, \ldots, S_{i_m}$ such that t is in S_i if and only if t is almost satisfied by a with respect to i. Suppose t is in S_i. If $a[i] = 0$ then t must contain the literal x_i, since flipping the ith bit from 0 to 1 causes t to become satisfied. Similarly, if $a[i] = 1$, then t must contain the literal $\overline{x_i}$. For notational convenience, define $S_i(x_i) = x_i$ if $a[i] = 0$, and $S_i(x_i) = \overline{x_i}$ if $a[i] = 1$. Thus, every term t in S_i contains the literal $S_i(x_i)$. Further, since F is read-k, for each equivalence class S_i, $|S_i|$ is at most k.

Consider a directed graph $G = (V, E)$ induced by the equivalence classes, with $V = \{S_{i_1}, S_{i_2}, \ldots, S_{i_m}\}$ and $E = \{\langle S_{i'}, S_i \rangle : \text{some term } t \in S_i \text{ contains the literal } \overline{S_{i'}(x_{i'})}\}$. (Note that $t \in S_i$ cannot contain the literal $S_{i'}(x_{i'})$ since this literal is negated in a, and t is almost satisfied by a with respect to a different literal).

Since each of the (at least one) terms in S_i contains the literal $S_i(x_i)$, and since F is read-k, there are at most $k-1$ other equivalence classes $S_{i'}$ containing a term which has the literal $\overline{S_i(x_i)}$. Thus, the outdegree of every vertex is at most $k-1$. Suppose G' is any subgraph of G containing μ vertices. Since there are at most $\mu(k-1)$ edges in G', there must be some vertex $S \in G'$ whose indegree + outdegree in G' is at most $2\mu(k-1)/\mu = 2(k-1)$. That is, S has at most $2(k-1)$ neighbors in the underlying undirected graph of G'.

Now assume, contrary to this lemma, that $|Y(a)|$ ($= m$, the number of vertices in G), is larger than $2kj$. Then there must be an independent set $V_{ind} \subseteq V$ of G of size at least $j+1$: Such an independent set V_{ind} can be constructed by first placing into V_{ind} some vertex S_i satisfying the indegree + outdegree bound above, and removing from G each of the at most $2(k-1)$ vertices adjacent to S_i in the underlying undirected graph. Then another vertex $S_{i'}$ is included in V_{ind}, its neighbors are removed, etc. After iteratively choosing j such vertices, (no two of which are adjacent in the underlying undirected graph), and removing their neighbors, we will have eliminated at most $j \cdot (2(k-1)+1) = 2kj - j < 2kj$ vertices. Thus, we can include at least one more vertex into the set V_{ind}, so that $|V_{ind}| \geq j+1$ as desired.

Let $I = \{i : S_i \in V_{ind}\}$, namely the set of indices of the at least $j+1$ equivalence classes with no edges between them in G. Consider a term t in some equivalence class S_i with $i \in I$. By definition of the equivalence classes, t is almost satisfied by a with respect to i, and thus *flip*(a, i) will satisfy term t. Since no other literal in t has index in I by definition of the set V_{ind}, *flip*(a, I) satisfies t as well. Thus *flip*(a, I) satisfies every term from each of the $j+1$ equivalence classes indexed by I, contradicting the assumption that F was a satisfy-j formula. ■

Definition 5 *A term $\tilde{t}$ is a p-variant of a term t if it contains all the literals of t and at most p additional literals.*

Corollary 6 *If a is a satisfying assignment for an RkSj DNF formula F satisfying the single term $t \in F$ then t is a $2kj$-variant of sensitive(a).*

Proof: First observe by Property 2 that *sensitive*$(a) \subseteq t$. Thus to prove this corollary it is sufficient to show that there are at most $2kj$ literals in $t -$ *sensitive*(a). Consider why a literal ℓ with index i might be in t but not in *sensitive*(a). By the definition of sensitive set this means that $F(\mathit{flip}(a,i)) = F(a)$ and in this case $F(\mathit{flip}(a,i)) = F(a) = 1$. But ℓ in t implies that $t(\mathit{flip}(a,i)) = 0$, so *flip*$(a,i)$ must satisfy some term other than t. Since t is the only term satisfied by a, all the terms satisfied by *flip*(a,i) must be almost satisfied by a with respect to i. This means $i \in Y(a)$, and by Lemma 4 there can be at most $2kj$ such values of i, hence at most $2kj$ corresponding literals ℓ in $t -$ *sensitive*(a). ∎

Definition 7 *Let $a, b \in \{0,1\}^n$. Then b is a d-*neighbor *of a if $b = \mathit{flip}(a, I)$ for some set I of cardinality at most d. (That is, if the Hamming distance between a and b is at most d.)*

Lemma 8 *If a is a satisfying assignment for an RkSj DNF formula F satisfying exactly the terms $T = \{t_1, t_2, \ldots, t_q\} \subseteq F$ then there exists a $(j-1)$-neighbor b of a such that some term in T is a $2kj$-variant of sensitive(b).*

Proof: Since F is satisfy-j and a is a satisfying assignment it must be the case that $1 \le |T| = q \le j$. Note by Corollary 6 that if $q = 1$, t_1 is a $2kj$-variant of *sensitive*(a), and since a is a 0-neighbor of itself, the lemma follows. To see that the lemma is true for $1 < q \le j$ we will show that in this case either some term satisfied by a is a $2kj$-variant of *sensitive*(a) or else there exists some i such that *flip*(a,i) satisfies a proper subset of T, and $F(\mathit{flip}(a,i)) = 1$. It then follows by induction that for some set of indices $I \subseteq \{1, \ldots, n\}$ with $|I| \le j - 1$, some term satisfied by a is a $2kj$-variant of *sensitive*$(\mathit{flip}(a, I))$. The assignment *flip*(a, I) is the $(j-1)$-neighbor b of a mentioned in the lemma.

If some term in T is a $2kj$ variant of *sensitive*(a) then we are done. Otherwise each term in T must contain more than $2kj$ literals not in *sensitive*(a). Let t be a term in T and let $L = \{\ell_1, \ell_2, \ldots\}$ be a set of more than $2kj$ literals in t that are not in *sensitive*(a). By Lemma 4, $|Y(a)| \le 2kj$, so there must be some literal $\ell \in L$ (say, with index i) such that there does not exist a term in F almost satisfied by a with respect to i. Consequently, *flip*(a,i) cannot satisfy any term not satisfied by a, and must therefore satisfy a proper subset of the terms satisfied by a since t will no longer be satisfied. Since $\ell \notin$ *sensitive*(a), *flip*(a,i) still satisfies at least one term. ∎

While the statement of Lemma 8 may appear somewhat confusing at first glance, if examined further we see that it immediately suggests an algorithm for learning RkSj DNF formulas (deterministically) where k and j are constants: First, conjecture the null DNF formula. On receiving a positive counterexample a, generate a collection H of terms, at least one of which is a term of the target DNF formula. To build the collection H, simply enumerate each $(j-1)$-neighbor b of a, and then include in H every $2kj$-variant of the sensitive set of each such b. By Lemma 8, H will necessarily contain some term of F that a satisfied. Of course, H may also contain many terms that are not in F (though only polynomially many, assuming k and j are constants). On seeing a negative counterexample, the algorithm simply removes from H any term that is satisfied, since these cannot be in F. Thus, on positive counterexamples, the algorithm adds to its hypothesis H at least one correct term not yet included, and at most a polynomial number of incorrect terms. On each negative counterexample, at least one of the incorrect terms is removed. It follows that the number of counterexamples that can be received (and hence the number of equivalence queries until the algorithm has produced a DNF logically equivalent to the target) is bounded by a polynomial in n and the number of terms of the target RkSj DNF formula.

This general approach of finding many possible terms from a positive counterexample a, and discarding the bad terms when seeing negative counterexamples, is a fairly standard approach used in many of the algorithms for learning subclasses of DNF formulas discussed in earlier sections. In the particular case of RkSj DNF, we need to develop additional technical lemmas in order to devise an algorithm that works for values of k and j that are not constant.

The following is an immediate corollary of Lemma 4.

Corollary 9 *For all a such that $F(a) = 0$, at most $2kj$ of the neighbors of a (in the cube $\{0,1\}^n$) are satisfying assignments of F.*

Proof: The neighbors of a are exactly $\mathit{flip}(a,1), \ldots, \mathit{flip}(a,n)$. Hence if $\mathit{flip}(a,i)$ is a satisfying assignment then $i \in Y(a)$. ■

Let $\mathcal{U}$ denote the uniform distribution, and let "$x \in \mathcal{U}$" denote that x is selected uniformly at random (in $\{0,1\}^n$). Also, let "true" be the "all 1's" concept.

Lemma 10 *If F is an RkSj DNF, and $F \not\equiv \mathsf{true}$, then* $\mathbf{Prob}_{x\in\mathcal{U}}[F(x) = 0] \geq 2^{-2kj}$

Proof: The proof uses an argument similar to one in [Sim83]. Think of the cube $\{0,1\}^n$ as a graph in which each vertex is represented by a n-bit string and there is an edge between any two vertices of Hamming distance 1. Consider the vertex-induced subgraph $G = (V, E)$, where V is the set of points $a \in \{0,1\}^n$ such that $F(a) = 0$. Since $F \not\equiv \textsf{true}$, we know that there is at least one vertex in V. We further know by Corollary 9 that the minimum degree of any vertex in G is at least $n - 2kj$.

A standard argument shows that any nonempty subgraph of $\{0,1\}^n$ with minimum degree at least d has at least 2^d vertices. For completeness we give the proof, which is by induction on n. The cases where $n = 1$ or $d = 0$ are easily verified. For the more general cases, notice that cutting the cube in any coordinate i leaves the two sub-cubes of dimension $n - 1$, each with minimum degree at least $d - 1$. This is because each vertex has at most one neighbor in the other sub-cube. So we simply choose i so that we have a non-empty subset of V in each sub-cube. Using the induction we get at least $2 \cdot 2^{d-1} = 2^d$ vertices.

This implies that $|V| \geq 2^{n-2kj}$, and therefore $\mathbf{Prob}_{x \in \mathcal{U}}[F(x) = 0] \geq 2^{-2kj}$ ■

Lemma 11 *Let F be an RkSj formula. Then F has at most kjq terms of length (number of literals) at most q.*

Proof: Consider a graph with one vertex for each term in F of length at most q and an edge between two vertices if their corresponding terms are not simultaneously satisfiable. In other words, two terms are connected exactly when one of the terms contains some variable x_i and the other contains $\overline{x_i}$. Since F is Read-k, the maximum degree of a vertex in this graph is $q(k-1)$. Therefore, if the graph has N vertices, it must have an independent set of size at least $N/(q(k-1)+1)$: such an independent set could be formed by repeatedly choosing a vertex to put in the set, and then eliminating from consideration all of the at most $q(k-1)$ neighbors of the vertex. On the other hand, the size of any independent set in this graph can be at most j since F is a Satisfy-j DNF (any independent set of vertices corresponds to terms that can all be simultaneously satisfied). Therefore N, the number of terms of length at most q, is at most $j(q(k-1)+1) \leq jkq$. ■

The next lemma is a variant of Lemma 8, which is easier to use in our arguments to follow.

Lemma 12 *If a is a satisfying assignment for an RkSj formula F, then either some term satisfied by a is a $4kj$-variant of sensitive(a) or else there is a set of literals U, such that $|U| \geq 2kj$ and for all $\ell \in U$, the assignment $b =$ flip(a, ℓ) is a positive example satisfying a strict subset of the terms of F satisfied by a.*

Proof: Let T be the set of terms satisfied by a. If some term in T is a $4kj$ variant of *sensitive*(a) then we are done. Otherwise, let t be any term in T, and let $W = \{w_1, w_2, \ldots, w_{4kj}, \ldots\}$ be the set of at least $4kj$ literals that are in T but are not in *sensitive*(a). By Lemma 4, for at least $2kj$ literals in W there does not exist a term in F almost satisfied with respect to it. So for at least $2kj$ literals ℓ, the assignment *flip*(a, ℓ) does not satisfy any term not satisfied by a. As such an assignment does not satisfy the term t, it satisfies a strict subset of T. ■

5 The Learning Algorithm

On a positive counterexample the algorithm (with high probability) adds to its hypothesis a polynomial number of terms one of which is an implicant of the target RkSj DNF formula. On a negative counterexample the algorithm deletes terms from the hypothesis that are satisfied by that example (and therefore are not implicants of the function). The number of queries will be bounded by a polynomial in n plus the number of terms in the target RkSj DNF formula. The enumerative approach discussed earlier results in a polynomial-time algorithm only when k and j are constants, because the number of neighbors and variant-terms to be enumerated is $O(n^{O(kj)})$. We avoid the enumerative approach, and obtain an algorithm tolerating larger (non-constant) values of k and j, by implementing a probabilistic search in the spirit of [BR92]. We show the learnability of RkSj DNF for $k \cdot j = O(\frac{\log n}{\log \log n})$.

Our algorithm has the following high level structure (see Figure 1), which we describe here. The low level of the algorithm is described in the next two sub-sections.

Let F be the target function and $F = t_1 \vee t_2 \vee \ldots \vee t_s$ its RkSj representation (there may be more than one such representation; in such a case we fix one of them just for the purpose of the analysis). The main tools in the algorithm are procedures *produce-terms* and *find-useful-example* that together, given a positive counterexample b to the current hypothesis, find a set of terms such that with high probability one of them (call it t) is an implicant of F (in fact, t will be a *prime* implicant), and is implied by some term t_i satisfied by b. Note that such a term t is satisfied by all the assignments that satisfy t_i, but not by any negative assignment of F. Thus finding t is as good as (or even better than) finding the term t_i.

The disjunction of all the terms found in this way is added to the hypothesis of the algorithm, with which we now ask an equivalence query (Step (2)). A negative counterexample allows us to throw out from the hypothesis terms that do not imply F. A positive counterexample satisfies a term $t_i \in F$ we have not yet found, and we use it to run again the procedures *find-useful-example* and *produce-terms*.

```
Algorithm Learn-RkSj:

1. $H \leftarrow \phi$

2. $b \leftarrow EQ(H)$

3. if $b =$'true' then stop and output $H$.

4. Else, if $b$ is a negative counterexample then for every term $t \in H$ such that $t(b) = 1$
   delete $t$ from $H$.

5. Else (if $b$ is a positive counterexample) then

    (a) $S \leftarrow$ find-useful-example$(b)$.   /* This procedure returns a set of examples. */

    (b) For each $a \in S$, $H \leftarrow H \cup$ produce-terms$(a)$.

6. GOTO 2
```

Figure 1: The Algorithm *Learn-RkSj*

5.1 Producing terms

We now describe the way the procedure *produce-terms* works given a positive example x of F. In our analysis, we will assume that example x has the property that some term t_1 of F satisfied by x is (1) not in our hypothesis and (2) is a $4kj$-variant of $sensitive(x)$. We call such an example x a *useful* example. If F is a disjoint DNF, then any positive counterexample x will be useful by the special case of Lemma 8 with $j = 1$. For general RkSj DNF, we use procedure *find-useful-example* (described in Section 5.2) to find an example with these properties. The procedure *produce-terms* is described in Figure 2, where $q = c \log^2 n$ for some constant c.

Our goal is to find an implicant of F that is implied by t_1. In fact, we will find a "small" collection of potential terms such that one of them will be such an implicant. We start by finding $sensitive(x)$, and we then fix those literals for the remainder of the procedure. After fixing (projecting onto) those literals, we still have an RkSj DNF, but t_1 has at most $4kj$ literals by our assumption. Let $t = \ell_1 \ell_2 \cdots \ell_r$ ($r \leq 4kj$) be some minimal (prime) implicant of F implied by t_1. Note that this means that for each $i \in \{1, 2, \ldots, r\}$ the term $t^{(i)} = \ell_1 \cdots \ell_{i-1} \overline{\ell_i} \ell_{i+1} \cdots \ell_r$ is *not* an implicant of F, because then we could remove ℓ_i from t and obtain an implicant, contradicting the primality of t.

The idea is that we create a set of literals L that with high probability has the following two properties: (1) L has all the literals of t, and (2) L has size poly(kj). This is done

Procedure *produce-terms*(x):

1. Find $sensitive(x)$ (using n membership queries).

2. $L \leftarrow \emptyset$.

3. Repeat the following $m_1 = 2^{6kj} \ln(12kjs/\delta)$ times:

   Pick a random y that agrees with $sensitive(x)$ (that is, for each literal in $sensitive(x)$ we take $y_i = x_i$ and for any other literal, not in $sensitive(x)$, the corresponding bit is selected randomly). If y is negative (a membership query) then find all literals ℓ satisfied by y (and not in $sensitive(x)$) such that *flip*(y, ℓ) is a positive example. Put all those literals into L.

4. If $|L| \leq kjq^2$, then for each subset L' of L of size at most $4kj$, add to the output the term $sensitive(x) \cup L'$.

Figure 2: The Procedure *produce-terms*

Procedure *find-useful-example*(x):

Do the following $m_2 = 2^{j+1} \log(3s/\delta)$ times and produce the union of the outputs.

1. Let $x^{(0)} = x$.

2. For $i = 1, 2, \ldots, m_3$, for $m_3 = \frac{n}{2k} \ln(2j2^j)$, let $x^{(i)}$ be a random positive neighbor of $x^{(i-1)}$.

3. Output $\{x^{(0)}, x^{(1)}, \ldots, x^{(m_3)}\}$.

Figure 3: The Procedure *find-useful-example*

in Step (3). If we have such a set then we are done: given the set L, we can look at all $\sum_{i=0}^{4kj} \binom{|L|}{i} = |L|^{O(kj)}$ small subsets (Step (4)), and one of them will be t.

Recall that s is the number of terms in the RkSj representation of F, and let $\delta > 0$ be a constant confidence parameter.

Claim 13 *If x is a useful example and t_1, t are as described above, then the set L produced in Step (3) of procedure produce-terms contains all the literals in t with probability at least $1 - \delta/3s$.*

Proof: Consider a literal ℓ_i in t. In a random y, there is a $1/2^r$ chance that literals $\ell_1, \ldots, \ell_r$ are set so that $t^{(i)}$ is satisfied. Also,

$$\mathbf{Prob}_{y \in \mathcal{U}}[y \text{ is negative} \mid y \text{ satisfies } t^{(i)}] \geq 1/2^{2kj}.$$

The reason is that if we project variables in t to satisfy $t^{(i)}$, then we are left with an RkSj DNF which is not identically true, since $t^{(i)}$ is not an implicant of F. So, we can apply Lemma 10. For such an example y, which is negative and satisfies $t^{(i)}$, we know that *flip*(y, ℓ_i) is a positive example (as it satisfies t) and hence ℓ_i will be inserted into L. So, the probability that a single y "discovers" ℓ_i is at least $1/2^{r+2kj} \geq 1/2^{6kj}$. A standard calculation shows that if we repeat the loop (Step (3)) at least $m_1 = 2^{6kj} \ln(12kjs/\delta)$ times then we find *all* $r \leq 4kj$ literals of t, with probability at least $1 - \delta/3s$. (Note that the number of repetitions, m_1, is polynomial as long as $kj = O(\log n)$.) ■

Claim 14 *Let $q = c \log^2 n$, where c is constant that depends on the constant δ, and assume that $kj = O(\log n)$. With probability at least $1 - \delta/3s$ we get $|L| \leq kjq^2$.*

Proof: Any literal found in Step (3) of *produce-terms* must be in *some* term. So, consider terms of size at most q. By Lemma 11 there are at most kjq such terms. The number of literals that those terms can contribute to L is at most kjq^2. Now consider a term of size greater than q. With high probability, it will not contribute *any* literals to L. The reason is that in order for such a term to "matter" (i.e. for some *flip*(y, ℓ) to satisfy that term) it must be that y satisfies all but one literal in that term. The chance that y does so is at most $n/2^q$ which, by the choice of q, is smaller than any polynomial fraction. Since the algorithm repeats in Step (3) a polynomial number of times (using here that $kj = O(\log n)$), throughout the entire algorithm, with high probability, *no* example *flip*(y, ℓ) satisfies *any* of those terms. By choice of c this probability can be made larger than $1 - \delta/3s$. ■

Notice that by Claims 13 and 14, with high probability at least one of the terms output in Step (4) of the procedure *produce-terms* is the desired term t, and in addition at most $(kjq^2)^{4kj}$ terms are output in Step (4) total. This number of terms is polynomial in n as long as $kj = O(\log n / \log \log n)$.

5.2 Finding a useful positive example.

The procedure *produce-terms* in the previous section requires a useful example x. Namely, some term t_1 of F is (1) not in our hypothesis and (2) is a $4kj$-variant of $sensitive(x)$. In this section we show how to produce, given a positive counterexample x to our hypothesis, a polynomial-sized set of examples such that with high probability at least one is useful in this sense. The procedure is described in Figure 3. The idea behind procedure *find-useful-example* is as follows. Suppose that our given positive example x satisfies some number of terms of F (none of them appears in our hypothesis). Lemma 12 states that either:

(A) All but $4kj$ of the literals in one of those terms are sensitive (i.e., x is already useful), or else,

(B) There exist at least $2kj$ literals $u_1, \ldots, u_{2kj}$ such that flipping u_i will cause the example to remain positive but satisfy a strict subset of those terms and no others.

We also know, by Lemma 4, that $|Y(x)| \leq 2kj$ and therefore there are at most $2kj$ literals that when flipped cause the example to satisfy some additional term.

What this means is that either example x is already useful, or else out of the n literals, at least $2kj$ are "good" in that flipping them makes our example satisfy a strict subset of the original set of terms, at most $2kj$ are "bad" in that flipping them makes the example satisfy some new term, and the rest are "neutral" in that either flipping them makes the example negative (which we will notice) or else flipping them does not affect the set of terms satisfied. So, as described in Figure 3 (Step (2)), let $x^{(i)}$ to be a random positive neighbor of $x^{(i-1)}$ in the boolean hypercube.

Claim 15 *With probability $\frac{1}{2^{j+1}}$ at least one of the examples $x, x^{(1)}, \ldots, x^{(m_3)}$ is useful.*

Proof: For the moment, consider the infinite sequence $x^{(1)}, x^{(2)}, \ldots$ of positive examples. If the example x was not already useful, let's say that we are "lucky" if we flip one of the "good" literals before we flip any of the "bad" literals in this experiment (i.e., we are lucky if we flip an arbitrary number of neutral literals followed by a good literal). Notice that flipping a "neutral" literal may change the sets of good and bad literals, but our bounds on the sizes of those sets remain. So, the probability we are lucky is at least 1/2. If this occurs, we then either have a useful positive example, or else our bounds on the sizes of the good and bad sets remain but the example satisfies at least one fewer term. Thus, with probability at least $(1/2^j)$, we continue to flip good literals before we flip any bad literals and we reach a useful example (since by Corollary 6 if the example satisfies only one term then it is useful).

Now for the finite sequence $x^{(1)}, \ldots, x^{(m_3)}$, we have to subtract the probability of being lucky j times, but not within the first m_3 trials. This event is included in the event "there were less than j flips of non-neutral literals within m trials". Let the latter event be E.

Consider m_3 as j blocks of $\frac{n}{2kj}\ln(2j2^j)$ trials each. The probability that we do not flip any non-neutral bit (or reach a useful example) in one block is at most $\frac{1}{2j2^j}$ (since the probability of getting a non-neutral bit is at least $\frac{2kj}{n}$ at each trial). The probability that we fail in any of the j blocks is therefore at most $\frac{1}{2^{j+1}}$. This is also a bound for the probability of the event E. Thus our total success probability is at least $\frac{1}{2^j} - \frac{1}{2^{j+1}}$. ∎

Claim 16 *With probability at least $1-\delta/3s$, some example in the set of examples produced by the procedure find-useful-example is useful.*

Proof: Procedure *find-useful-example* repeats the experiment described in Claim 15 for $m_2 = 2^{j+1}\ln(3s/\delta)$ times. Thus, with probability at least $1-\delta/3s$, the experiment is successful at least once. ∎

5.3 Final analysis

Claim 17 *With probability at least $1-\delta$, the algorithm Learn-RkSj uses $O(snm_1m_2m_3)$ membership queries and $O(sm_2m_3(kjq^2)^{4kj})$ equivalence queries.*

Proof: By Claim 16, each time that procedure *find-useful-example* is invoked (on a positive counterexample), we get a useful example with probability at least $1-\delta/3s$. By Claim 13 and Claim 14, if we have a useful example then we find a term t that satisfies the original counterexample with probability at least $1-2\delta/3s$. All together we find a good term t with probability at least $1-\delta/s$. Therefore, if we get s positive counterexamples, with probability at least $1-\delta$, we find prime implicants for all the terms in F, and therefore the hypothesis is logically equivalent to F.

We now count the number of queries used in this case. In each call to *produce-terms* we make n membership queries to find $sensitive(x)$, one membership query for each y, and n additional queries for each y that is negative. All together the procedure *produce-terms* makes $O(nm_1)$ membership queries to create the set of literals L. It then produces (in Step (4)) $O((kjq^2)^{4kj})$ terms (or none, in the unlikely event that $|L| > kjq^2$). The number of calls to the procedure *produce-terms* is bounded by the number of positive counterexamples that we get in the algorithm (which is bounded by s) multiplied by the number of examples produced by the procedure *find-useful-example* on each such counterexample (this is bounded by m_2m_3). Therefore, in total we make $O(snm_1m_2m_3)$ membership queries and produce $O(sm_2m_3(kjq^2)^{4kj})$ terms. This immediately gives a bound on the number of negative counterexamples. Hence, the number of equivalence queries is $O(sm_2m_3(kjq^2)^{4kj})$. Note that just by the Read-k property of F we have $s \leq kn$. A tighter bound of $s = O(\sqrt{k^2jn})$ is given by [Mat93]. ■

Claim 18 *Let $\delta > 0$ be a constant and $kj = O(\frac{\log n}{\log\log n})$. The algorithm Learn-RkSj runs in time poly(n) and finds a function logically equivalent to F with probability at least $1-\delta$.*

Proof: By Claim 17, the algorithm succeeds with probability at least $1-\delta$. The polynomial bound then follows, noting that m_1, m_2 and m_3 are all polynomial for $kj = O(\log n)$ and that $(kjq^2)^{O(kj)}$ is polynomial for $kj = O(\log n/\log\log n)$ (as q is *polylog*(n)). ■

So far we considered only δ which is a fixed constant (this is used in Claim 14 when we say that c is a constant). To generalize to arbitrary δ (e.g, δ can be a function of n), we use a standard technique as follows:

Theorem 19 *Let F be an RkSj formula, $kj = O(\frac{\log n}{\log\log n})$ and $\delta > 0$. There is an algorithm that runs in time poly$(n, \log\frac{1}{\delta})$ and finds a function logically equivalent to F with probability at least $1-\delta$.*

Proof: We run the algorithm *Learn-RkSj* $\ln\frac{1}{\delta}$ times. Every time we stop after using the number of queries stated in Claim 17, and use an equivalence query to decide if we need to run further. Clearly, the overall algorithm runs in *poly*$(n, \log\frac{1}{\delta})$ and succeeds with probability at least $1-\delta$. ■

As a last remark we note that it may be that the method we use here can be adapted for $kj = O(\log n)$. The only part of the algorithm that does not allow for $kj = O(\log n)$ is the procedure *produce-terms* and in particular Step (4) that produces $O((\log n)^{O(kj)})$ terms. Finding a better sampling method or a better way to produce the terms out of the set L might solve this problem.

6 Read-2 Disjoint DNF does not have small CNF

We present a family $\{F_n\}$ of functions that have short (*poly*(n)) R2D DNF formulas but require CNF formulas of size $2^{\Omega(\sqrt{n})}$. Note that this also implies that the size of any decision tree for these functions must also be super-polynomial, even without restricting the number of variable occurrences. Bshouty [Bsh95] gives an algorithm for learning functions (such as decision trees) which have both small DNFs **and** small CNFs. Our family of functions $\{F_n\}$ show that this result does not apply here as the size of the CNFs for even read-2 disjoint DNFs may be super-polynomial. It remains open, however, whether techniques similar to those used in [Bsh95] can be applied to yield a comparable, or stronger result. Such a result would rely on showing that every RkSj DNF has a small "monotone basis"; we suspect this is not the case as even read-twice DNFs (though not disjoint) require a exponentially large monotone basis (Bshouty, private communication).

Theorem 20 *There is a family of functions $\{F_n\}$ with polynomial-sized R2D representations but whose CNF representations require at least $2^{\sqrt{2n}} - 2(\sqrt{2n}+1)$ clauses.*

Proof: Let $n = \binom{m+1}{2}$. We define a function F_n with $m+1$ terms each of length m on n variables $x_{i,j}$ where $1 \le i < j \le m+1$. The R2D representation of the function is $t_1 \vee t_2 \vee \ldots \vee t_{m+1}$, where the term t_q includes all the literals $x_{q,j}$ for $j > q$, and all the literals $\overline{x_{j,q}}$ for $j < q$. The idea is that the variable $x_{i,j}$ appears only in the i-th term and the j-th term, and is "responsible" for the disjointness of these two terms (since the variable appears negated in one term, and un-negated in the other). For example, for $m = 3$ the function is: $x_{12}x_{13}x_{14} \vee \overline{x_{12}}x_{23}x_{24} \vee \overline{x_{13}}\,\overline{x_{23}}x_{34} \vee \overline{x_{14}}\,\overline{x_{24}}\,\overline{x_{34}}$. The main step in the proof is the following claim:

Claim 21 *Every clause in a CNF representation of F_n must have $m+1$ literals.*

Given this claim, the following counting argument shows that the CNF must have a large number of clauses: The number of assignments satisfied by F_n is exactly $(m+1)2^{n-m}$, since each of the $m+1$ terms satisfies 2^{n-m} assignments (it determines the value of m out of the n variables) and the representation is disjoint. This implies that the number of assignments unsatisfied by F_n is $2^n(1-(m+1)/2^m)$. Since a clause of length $\ge m+1$ falsifies at most 2^{n-m-1} assignments, we need at least $2^{m+1}(1-(m+1)/2^m) = 2^{m+1} - 2(m+1)$ clauses to falsify all the unsatisfying assignments of F_n. As $m^2 \le 2n \le (m+1)^2$ the theorem follows.
Proof of the Claim:
Assume by way of contradiction that there is a clause of length $\alpha < m+1$ in a CNF representation for F_n. Let $C = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_\alpha)$ be that clause. We show that there exists an assignment a such that $C(a) = 0$ (and hence the value of the CNF formula on a is 0) and $F_n(a) = 1$, contradicting the assumption that C is a clause in a CNF representation of F_n.

To construct a notice that since every *literal* appears exactly once in the R2D representation, fixing the values of $\alpha \le m$ variables can falsify at most m terms in the DNF representation, so there is still a term t_C that can be satisfied. Therefore, we can define a to be the assignment in which all the literals in C are set to zero, all the literals in t_C are set to one, and all other literals are set arbitrarily. We thus have that $C(a) = 0$ but $t_C(a) = 1$ (and hence $F_n(a) = 1$), a contradiction. ■

7 Bibliographic Remarks

The work presented here appeared in preliminary forms as [AP92] and [BKK$^+$94]. The algorithm for constant k and j is from [AP92] and the extension to $k \cdot j = O(\log n / \log\log n)$ from [BKK$^+$94]. The family of functions given in Section 6 was defined in [AP92], where it was shown that the class R2D is not included in Read-k decision trees. This result was later strengthened in [Aiz93] to show that the family requires decision trees of size 2^{n-2}. Its current form is from [BKK$^+$94].

References

[AFP92] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of Horn clauses. *Machine Learning*, 9:147–164, 1992.

[AHK93] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. *J. ACM*, 40:185–210, 1993.

[AHP92] H. Aizenstein, L. Hellerstein, and L. Pitt. Read-thrice DNF is hard to learn with membership and equivalence queries. In *Proc. of the 33rd Symposium on the Foundations of Comp. Sci.*, pages 523–532. IEEE Computer Society Press, Los Alamitos, CA, 1992. A revised manuscript with additional results will appear in *Computational Complexity*, as Aizenstein, Hegedus, Hellerstein, and Pitt, "Complexity Theoretic Hardness Results for Query Learning".

[Aiz93] H. Aizenstein. *On the Learnability of Disjunctive Normal Form Formulas and Decision Trees.* PhD thesis, Dept of Computer Science, University of Illinois, 1993. Technical report UIUCDCS-R-93-1813, June, 1993, Urbana, Illinois.

[AK95] D. Angluin and M. Kharitonov. When won't membership queries help? *Journal of Computer and System Sciences*, 50, 1995. Special issue for the 23rd Annual ACM Symposium on Theory of Computing.

[Ang87] D. Angluin. Learning k-term DNF formulas using queries and counterexamples. Technical Report YALEU/DCS/RR-559, Department of Computer Science, Yale University, August 1987.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[Ang90] D. Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.

[AP91] H. Aizenstein and L. Pitt. Exact learning of read-twice DNF formulas. In *Proceedings of the IEEE Symp. on Foundation of Computer Science*, number 32, pages 170–179, San Juan, 1991.

[AP92] H. Aizenstein and L. Pitt. Exact learning of read-k disjoint DNF and not-so-disjoint DNF. In *Proceedings of the Annual ACM Workshop on Computational Learning Theory*, pages 71–76, Pittsburgh, Pennsylvania, 1992. Morgan Kaufmann.

[AP95] H. Aizenstein and L. Pitt. On the learnability of disjunctive normal form formulas. *Machine Learning*, 19(3):183–208, 1995.

[BFJ$^+$94] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. Rudich. Weakly learning DNF and characterizing statistical query learning using fourier analysis. In *Proceedings of Twenty-sixth ACM Symposium on Theory of Computing*, 1994.

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth International Group, 1984.

[BKK$^+$94] A. Blum, R. Khardon, A. Kushilevitz, L. Pitt, and D. Roth. On learning read-k satisfy-j DNF. In *Proceedings of the Annual ACM Workshop on Computational Learning Theory*, pages 110–117, 1994.

[BR92] A. Blum and S. Rudich. Fast learning of k-term DNF formulas with queries. In *Proceedings of Twenty-Fourth ACM Symposium on Theory of Computing*, pages 382–389. ACM, 1992.

[BS90] A. Blum and M. Singh. Learning functions of k terms. In *Proceedings of COLT '90*, pages 144–153. Morgan Kaufmann, 1990.

[Bsh95] N. H. Bshouty. Exact learning boolean functions via the monotone theory. *Information and Computation*, 123(1):146–153, 1995. Earlier version appeared in Proc. 34rd Ann. IEEE Symp. on Foundations of Computer Science, 1993.

[Han91] T. Hancock. Learning 2μ DNF formulas and $k\mu$ decision trees. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 199–209, Santa Cruz, California, August 1991. Morgan Kaufmann.

[Jac94] J. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. In *Proceedings of the IEEE Symp. on Foundation of Computer Science*, number 35, pages 42–53, Santa Fe, NM., 1994.

[Kha94] R. Khardon. On using the Fourier transform to learn disjoint DNF. *Information Processing Letters*, 49(5):219–222, March 1994.

[KLPV87] M. Kearns, M. Li, L. Pitt, and L. Valiant. On the learnability of Boolean formulae. In *Proc. 19th Annu. ACM Sympos. Theory Comput.*, pages 285–294. ACM Press, New York, NY, 1987.

[KM93] E. Kushilevitz and Y. Mansour. Learning decision trees using the fourier spectrum. *SIAM Journal of Computing*, 22(6):1331–1348, 1993. Earlier version appeared in the 23rd Annual ACM Symposium on Theory of Computing, 1991.

[KR93] E. Kushilevitz and D. Roth. On learning visual concepts and DNF formulae. In *Proceedings of the ACM Workshop on Computational Learning Theory '93*, pages 317–326. Morgan Kaufmann, 1993. To apppear in Machine Learning, 1996.

[Mat93] S. Matar. Learning with minimal number of queries. Master's thesis, University of Alberta, Canada, 1993.

[PR94] K. Pillapakkamnatt and V. Raghavan. On the limits of proper learnability of subsets of DNF formulas. In *Proceedings of the Annual ACM Workshop on Computational Learning Theory*, pages 118–129, 1994. To apppear in Machine Learning, 1996.

[PR95] K. Pillapakkamnatt and V. Raghavan. Read twice DNF formulas are properly learnable. *Information and Computation*, 122(2):236–267, 1995.

[PV88] L. Pitt and L. Valiant. Computational limitations on learning from examples. *J. ACM*, 35:965–984, 1988.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Qui93] J. R. Quinlan. *C4.5: Programs for machine learning.* Morgan Kaufmann, 1993.

[Sim83] H. U. Simon. A tight $\Omega(\log\log n)$-bound on the time for parallel RAM's to compute nondegenerate boolean functions. In *ICALP*, number 158 in Lecture Notes in Computer Science, pages 439–444. Springer-Verlag, 1983.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[Val85] L. G. Valiant. Learning disjunctions of conjunctions. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence, vol. 1*, pages 560–566, Los Angeles, California, 1985. International Joint Committee for Artificial Intelligence.

8 Appendix: Lemma 4 is Tight

Lemma 4 plays a central role in our analysis. It gives a bound on the number of terms which are almost satisfied by an assignment a, in terms of k and j. It is natural to ask how good is this bound. The following example, due to Galia Givaty (private communication), shows that it is essentially optimal (up to constants). More precisely, for any values of k and j we give a function $F_{k,j}$ with $k \cdot j$ variables $x_0, x_2, \ldots, x_{kj-1}$, and an assignment a such that $|Y(a)| = k \cdot j$. The function $F_{k,j}$ has kj terms:

$$t_i = \overline{x_i} x_{i+1} \ldots x_{i+k-1 \bmod kj} \qquad (0 \le i \le kj-1)$$

It can be easily verified that the function is Read-k (the variable x_i appears only in the terms $t_{i-k+1 \bmod kj}, \ldots, t_i$). It is also Satisfy-j as if an assignment satisfies a term t_i it cannot satisfy any of the $k-1$ terms $t_{i-k+1 \bmod kj}, \ldots, t_{i-1 \bmod kj}$ (as they all contain the variable x_i while t_i contains $\overline{x_i}$). Now, let a be the all "1" assignment. Each term t_i is almost satisfied by a with respect to index i. Hence, $|Y(a)| = kj$.