

Learning with Errors in Answers to Membership Queries

Laurence Bisht

Nader H. Bshouty *

Lawrance Khoury

Department of Computer Science

Technion, 32000

Haifa, Israel

bisht,bshouty,khoury@cs.technion.ac.il

Abstract

We study the learning models defined in [AKST97]: Learning with equivalence and limited membership queries and learning with equivalence and malicious membership queries.

We show that if a class of concepts that is closed under projection is learnable in polynomial time using equivalence and (standard) membership queries then it is learnable in polynomial time in the above models. This closes the open problems in [AKST97].

Our algorithm can also handle errors in the equivalence queries.

1 Introduction

We study the exact learning model defined by Angluin et. al. [AKST97] from equivalence and membership queries with ℓ errors or omissions in answers to membership queries. The *limited* membership query may give a special “I don’t know” answer, while a *malicious* membership query may give a wrong answer. Any class of concepts learnable in polynomial time using equivalence and malicious membership queries is learnable in polynomial time using equivalence and limited membership queries. Angluin et. al. [AKST97] stated the converse as an open problem and they showed that the classes of monotone DNF, DFA, and decision trees are learnable with equivalence and malicious membership queries.

In this paper, we show that for concepts that are closed under projection both models are equivalent to the exact learning model without errors. That is, if a class C is closed under projection and is learnable from membership and equivalence queries then C is learnable from malicious (resp. limited) membership and equivalence queries. We note that closure under projection is, in some sense, not a constraint because all the classes considered in the literature are classes that are closed under projection.

Our technique can also handle errors in answering equivalence queries. We show that if a class is closed under projection and is learnable from equivalence queries only then it is learnable from malicious equivalence queries only, i.e., equivalence query that returns ℓ wrong answers.

*This research was supported by the fund for promotion of research at the Technion.

We also consider limited membership query $\text{LMQ}_{\mathcal{K}}$ for some set of sub-domains \mathcal{K} . This query returns correct answers for all assignments that are in some sub-domain $K \in \mathcal{K}$ chosen by some adversary. We show that if \mathcal{K} (as concept class of boolean functions) is learnable as disjoint DNF from membership and equivalence queries then learning with limited membership queries $\text{LMQ}_{\mathcal{K}}$ and equivalence queries is equivalent to learning with membership and equivalence queries.

There have been very few works on omissions and errors in the exact learning model. Angluin and Slonim [AS94] introduced the exact learning model with Randomly Fallible Teachers. In this model, the teacher with probability p answers “I don’t know” to the membership query¹. Very few classes are known to be learnable in this model [GM92, BO01, BE02]. Nothing is known to be learnable in the exact learning model where the teacher lies on answering membership queries with a constant probability p . On the other hand, many results are known in the PAC learning model [V84] when the teacher lies on answering membership queries with some probability [GKS93, RR95, JSS99, BF02].

In section 2 we give some preliminary results. In section 3 we define the learning models. In section 4 we show a general technique for learning classes that are closed under projection that will be used in the sequel sections. In section 5 we study learning with limited membership queries. In section 6 we study learning with malicious membership queries and malicious equivalence queries with ℓ errors and show that it is equivalent to learning with membership and equivalence queries.

2 Preliminaries

A *concept class* is $C = \cup_{n>0} C_n$ where each C_n is a set of boolean functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$. A *partial assignment* p is (p_1, \dots, p_n) where $p_i \in \{0, 1, x_i\}$. We say that a concept class C is *closed under projection* if for every $f \in C$ and every partial assignment p , $f(p) \in C$. We will sometimes write f_p for $f(p)$. We say that C is *closed under perfect projection* if for any $f(x_1, \dots, x_n) \in C$ we have $f(x_1, \dots, x_{n-1}, 0) \in C$ and $f(x_1, \dots, x_{n-1}, 1) \in C$. It is clear that if C is closed under projection then it is closed under perfect projection.

For $\lambda \in \{0, 1\}$ and a variable y we define $y^\lambda = y$ if $\lambda = 1$ and $y^\lambda = \bar{y}$ if $\lambda = 0$. For a partial assignment $p = (p_1, \dots, p_n)$ and an assignment $a = (a_1, \dots, a_n)$ we write $p(a)$ for the assignment $b = (b_1, \dots, b_n)$ where $b_i = a_i$ if $p_i = x_i$ and $b_i = p_i$ otherwise.

A set of terms $\mathcal{T} = \{T_1, T_2, \dots, T_m\}$ is called *complete set of terms* if for every assignment $a \in \{0, 1\}^n$ exactly one term T_i satisfies a , i.e., $T_i(a) = 1$ for exactly one i . For example: The set $\mathcal{T} = \{x_1 x_3, x_1 \bar{x}_3, \bar{x}_1 x_2, \bar{x}_1 \bar{x}_2\}$ is a complete set of terms. Define for an assignment $a \in \{0, 1\}^n$ the terms: $T_{a,n} = x_n^{1-a_n}$, $T_{a,0} = x_n^{a_n} \dots x_1^{a_1}$ and for $0 < k < n$,

$$T_{a,k} = x_n^{a_n} \dots x_{k+1}^{a_{k+1}} x_k^{1-a_k}.$$

We now show

Fact 1 *For any assignment $a \in \{0, 1\}^n$ the set*

$$\mathcal{T}^a = \{T_{a,k} \mid k = 0, 1, 2, \dots, n\}$$

¹In all the models we assume that the teacher is persistent, i.e., if it queried more than once on the same assignment, it will always return the same answer

is a complete set of terms.

Proof. Notice that $T_{a,k_1} \wedge T_{a,k_2} = 0$ for $k_1 \neq k_2$ and therefore it is enough to show that every assignment satisfies at least one term.

Let $d \in \{0, 1\}^n$. If $d = a$ then $T_{a,0}(d) = 1$. If $d \neq a$ then let k be the maximal integer such that $d_n = a_n, d_{n-1} = a_{n-1}, \dots, d_{k+1} = a_{k+1}$ and $d_k \neq a_k$. Then, $T_{a,k}(d) = 1$. \square

A decision tree is a rooted binary tree whose internal nodes are labeled with variables $\{x_1, x_2, \dots, x_n\}$ and whose leaves are labeled with constants $\{0, 1\}$. Each internal node has precisely two outgoing edges, one labeled with 0 and the other labeled with 1. A decision tree computes a Boolean function from $\{0, 1\}^n$ to $\{0, 1\}$ in the following natural way. Given an assignment $a \in \{0, 1\}^n$, the computation starts at the root node. At each node v , if it is labeled with x_i then the computation takes the edge labeled with a_i out to the next node. The computation stops at a leaf node and outputs the label at this node. The *size of a decision tree* D is the number of leaves in D .

A disjoint DNF is sub-class of DNF that contains all the functions that can be represented by a DNF formulae whose terms are “disjoint”, that is, every assignment a satisfies at most one term. This is equivalent to that the conjunction of every two terms in the DNF is 0. The *size of a (disjoint)DNF* function f is the number of terms in f . It is easy to show that any decision tree D with s leaves can also be represented by a disjoint DNF with at most s terms. We can correspond to each leaf in the tree D a term (as described below). The disjunction of all terms that correspond to the leaves in D that are labeled with 1 forms a disjoint DNF that is logically equivalent to D .

Complete set of terms can be built using any decision tree D with non-labeled leaves. Let v be a leaf of D and let $v_1, v_2, \dots, v_m = v$ be the path from the root of D to v . Let x_{i_j} be the label of v_j and ψ_j is the label of the edge (v_j, v_{j+1}) , $j < m$. Then the term

$$T[v] = x_{i_1}^{\psi_1} x_{i_2}^{\psi_2} \dots x_{i_{m-1}}^{\psi_{m-1}}$$

is called the term that corresponds to the leaf v in D . This term satisfies: $T[v](a) = 1$ if and only if the computation of $D(a)$ stops at leaf v . Each leaf in the tree D corresponds to a term and each assignment belongs to (respectively, satisfies) exactly one leaf (respectively, exactly one term). We denote by $\mathcal{T}(D)$ the set of terms that correspond to the leaves of D . We now show

Fact 2 *For every decision tree D , the set $\mathcal{T}(D)$ is a complete set of terms.*

Proof. Every assignment a corresponds to exactly one leaf and therefore satisfies only the term that corresponds to this leaf. \square

We will also call $\mathcal{T}(D)$, the complete set of terms that corresponds to the decision tree D . For an assignment $a \in \{0, 1\}^n$ define D_a the following decision tree:

If $(x_n = \overline{a_n})$ then \circ elseif $(x_{n-1} = \overline{a_{n-1}})$ then \circ elseif \dots elseif $(x_1 = \overline{a_1})$ then \circ else \circ

where \circ is an empty leaf. Then

$$\mathcal{T}(D_a) = \mathcal{T}^a.$$

The decision tree (if exists) that corresponds to a complete set of terms \mathcal{T} is a decision tree D such that $\mathcal{T}(D) = \mathcal{T}$. Not every complete set of terms corresponds to a decision tree. For

example the set $\mathcal{T}_1 = \{\bar{x}_1\bar{x}_2\bar{x}_3, x_1x_2x_3, x_1\bar{x}_3, \bar{x}_1x_2, \bar{x}_2x_3\}$ is a complete set of terms that cannot be generated from a decision tree.

Let $S = \{a^{(1)}, \dots, a^{(l)}\} \subseteq \{0, 1\}^n$ be a set of assignments. For a term T , let $A(T) \triangleq \{a | T(a) = 1\}$. We say that a complete set of terms \mathcal{T} *isolates* S if for every term $T \in \mathcal{T}$ either $A(T) \subseteq S$ or $A(T) \subseteq \bar{S} = \{0, 1\}^n \setminus S$. For example, if $n = 3$ then the above complete set of terms \mathcal{T}_1 isolates the set $S = \{(000), (100), (110), (111)\}$. Notice that \mathcal{T}^a isolates $S = \{a\}$.

We now prove the following

Lemma 3 *Let $S \subseteq \{0, 1\}^n$ and let χ_S be the characteristic boolean function of S , i.e., $\chi_S(a) = 1$ if $a \in S$ and $\chi_S(a) = 0$ otherwise. Then*

1. *There is a complete set of terms of size s that corresponds to a decision tree that isolates S if and only if there is a decision tree of size s for χ_S .*
2. *There is a complete set of terms of size s that isolates S if and only if there is a disjoint DNF for χ_S of size s_1 and a disjoint DNF for $\bar{\chi}_S$ of size s_2 and $s = s_1 + s_2$.*

Proof. Let \mathcal{T} be a complete set of terms of size s that isolates S that corresponds to a decision tree D . That is, $\mathcal{T} = \mathcal{T}(D)$ and for every $T \in \mathcal{T}$ either $A(T) \subseteq S$ or $A(T) \subseteq \bar{S}$. For every leaf in D we label this leaf with 1 if the corresponding term T of this leaf satisfies $A(T) \subseteq S$ and 0 otherwise. Since

$$\bigcup_{T \in \mathcal{T}(D)} A(T) = \{0, 1\}^n,$$

and for every $T \in \mathcal{T}(D)$ either $A(T) \subseteq S$ or $A(T) \subseteq \bar{S}$ we have

$$\bigcup_{T \in \mathcal{T}(D), A(T) \subseteq S} A(T) = S, \tag{1}$$

and therefore D with the labeled leaves is a decision tree of size s for χ_S .

On the other hand, if D is a decision tree for χ_S then (1) is true and therefore, $\mathcal{T}(D)$ is a complete set of terms that isolates S . This completes the proof for 1.

To prove 2 let \mathcal{T} be a complete set of terms that isolates S . Then it is easy to see that

$$\bigvee_{T \in \mathcal{T}, A(T) \subseteq S} T = \chi_S, \text{ and } \bigvee_{T \in \mathcal{T}, A(T) \subseteq \bar{S}} T = \bar{\chi}_S.$$

On the other hand, let $\chi_S = T_1 \vee T_2 \vee \dots \vee T_{s_1}$ and $\bar{\chi}_S = T'_1 \vee T'_2 \vee \dots \vee T'_{s_2}$ be disjoint DNF. Let $\mathcal{T} = \{T_i | i \leq s_1\} \cup \{T'_j | j \leq s_2\}$. Then, $\bigvee_i T_i \vee \bigvee_j T'_j = \chi_S \vee \bar{\chi}_S = 1$, $A(T_i) \subseteq S$, $A(T'_j) \subseteq \bar{S}$. Also $T_i \wedge T'_j \Rightarrow \chi_S \wedge \bar{\chi}_S = 0$ and therefore \mathcal{T} is a complete set of terms that isolates S . \square

In the Appendix, we show that there is a polynomial gap between the minimal size of complete set of terms that isolates S and the minimal size of a complete set of terms that corresponds to a decision tree that isolates S . We show

Lemma 4 *There is a set of assignments S that has a complete set that isolates it of size s but any complete set that isolates S that corresponds to a decision tree is of size at least*

$$s^{\frac{\log 6}{\log 5} - o(1)} \geq s^{1.11328275}.$$

We say that a complete set of terms \mathcal{T} is *perfect complete set* if $\mathcal{T} = \{1\}$ or for each term $T \in \mathcal{T}$ there is an assignment $a \in \{0, 1\}^n$ and $0 \leq k \leq n$ such that $T = T_{a,k}$.

We now show

Lemma 5 *We have*

1. *Every perfect complete set corresponds to some decision tree. That is, for every perfect complete set of terms \mathcal{T} there is a decision tree D such that $\mathcal{T}(D) = \mathcal{T}$.*
2. *For every set of assignments $S \subseteq \{0, 1\}^n$, where $|S| > 1$, there is a perfect complete set of terms of size at most $n|S|$ that isolates S .*

Proof. We prove (1) by induction on n . Since \mathcal{T} is perfect, it is either $\mathcal{T} = \{1\}$ or each term in \mathcal{T} contains either x_n or \bar{x}_n . If $\mathcal{T} = \{1\}$ then the empty leaf tree $D = \circ$ satisfies $\mathcal{T}(D) = \mathcal{T}$. If $\mathcal{T} \neq \{1\}$ then since there is a term T that satisfies the zero vector $\mathbf{0}$, i.e., $T(\mathbf{0}) = 1$, and a term that satisfies the one vector $\mathbf{1}$, there is at least one term in \mathcal{T} that contains the literal x_n and one term that contains the literal \bar{x}_n . We define a decision tree D with a label x_n in its root. This splits \mathcal{T} into two perfect complete sets over $n - 1$ variables, for $x_n = 1$ we have \mathcal{T}_1 the set of terms in \mathcal{T} that contains x_n with x_n removed, and, for $x_n = 0$ we have \mathcal{T}_0 the set of terms in \mathcal{T} that contains \bar{x}_n with \bar{x}_n removed. Obviously, \mathcal{T}_0 and \mathcal{T}_1 are perfect complete sets over $n - 1$ variables. Now by the induction hypothesis \mathcal{T}_0 and \mathcal{T}_1 corresponds to decision trees D_0 and D_1 . We place D_0 as the 0-son of the root of D and D_1 as the 1-son and get a decision tree for \mathcal{T} . We can write this decision tree as: $Tree(n, \{1\}) = \circ$ and if $\mathcal{T} \neq \{1\}$ then

$$Tree(n, \mathcal{T}) := \text{If } (x_n = 1) \text{ then } Tree(n - 1, \mathcal{T}_1) \text{ else } Tree(n - 1, \mathcal{T}_0)$$

where for $\xi \in \{0, 1\}$,

$$\mathcal{T}_\xi = \{T \in \mathcal{T} \mid x_n^\xi \cdot T \in \mathcal{T}\}.$$

To prove (2) we build a decision tree for χ_S as follows: If $S = \emptyset$ then the tree is $D = \circ$. If $S \neq \emptyset$ then we label the root of the tree with x_n . For $x_n = 0$ (respectively, $x_n = 1$) we recursively build a decision tree for $S_0 = \{a \in S \mid a_n = 0\}$ (respectively, $S_1 = \{a \in S \mid a_n = 1\}$). If $s(n, S)$ is the size of the tree then $s(n, \emptyset) = 1$. If $S_i = \emptyset$ for some $i \in \{0, 1\}$ then $s(n, S) = s(n - 1, S_{1-i}) + 1$. If both are non-empty then $s(n, S) = s(n - 1, S_0) + s(n - 1, S_1)$. Now it is easy to prove by induction that $s(n, S) = n + 1$ for $|S| = 1$ and $s(n, S) \leq |S|n$ for $|S| > 1$. \square

In the Appendix we show that the above bound is tight even for (non-perfect) complete sets. We prove

Lemma 6 *There is a set of assignments $S \subset \{0, 1\}^n$ of polynomial size (in n) such that any complete set of terms for S is of size $\Omega(n|S|)$.*

3 The Exact Learning Model

In this paper we try to exactly identify a target function hidden by a teacher using queries. Although the functions we consider are boolean functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$, our technique can also be applied to other boolean function domains.

Let C_n be a class of boolean functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that can be represented as a boolean circuit and $C = \cup_{n \geq 0} C_n$. The *size* of $f \in C$ is the minimal number of nodes in any circuit that represents f . We will write $\text{size}(f)$ for the size of f and $n(f)$ for the dimension of the domain of f .

In learning, a *learner* has access to a set of *queries* Q for some hidden function $f \in C$. The goal of the learner is to run in polynomial time in $n(f)$, $\text{size}(f)$, ask queries from Q and output a polynomial size circuit that is equivalent to f .

We will consider the following queries

1. A *membership query* $\text{MQ}(a)$ for f , for some $a \in \{0, 1\}^n$, returns $f(a)$.
2. An *equivalence query* $\text{EQ}(h)$ for f , for some polynomial size circuit h , either return “YES”, meaning that h is logically equivalent to f (that is, for every $x \in \{0, 1\}^n$ we have $h(x) = f(x)$), or “NO”, meaning that h and f are not logically equivalent, together with a counterexample (that is, an assignment b such that $h(b) \neq f(b)$).

We say that a learning algorithm \mathcal{A} (exactly) *learns* C from a set of queries Q if for any target function $f \in C$ the algorithm \mathcal{A} uses queries from Q and outputs a function h that is logically equivalent to f . We say that a learning algorithm \mathcal{A} (exactly) *learns* C in polynomial time from a set of queries Q if for any target function $f \in C$ the algorithm \mathcal{A} uses queries from Q and after polynomial time in $n(f)$ and $\text{size}(f)$ outputs a function h that is logically equivalent to f . For a class H , we say that a learning algorithm \mathcal{A} *learns* C as H from a set of queries Q if it learns C where the hypotheses used for the equivalence queries (if it is in Q) and the output hypothesis are from H .

We will also consider membership and equivalence queries that do not answer or lie on some sub-domain of the function.

Let \mathcal{K} be a set of sub-domains of $\{0, 1\}^n$. In the *limited membership queries* $\text{LMQ}_{\mathcal{K}}$ an adversary chooses some $K \in \mathcal{K}$ and then $\text{LMQ}_{\mathcal{K}}(a) = f(a)$ if $a \in K$ and $\text{LMQ}_{\mathcal{K}}(a) = \text{“I don’t know”}$ if $a \notin K$. In this model we define a hypothesis to be “non-strict” correct if it agrees with the target concept on all examples except possibly ones for which the limited membership query answers “I don’t know”. Thus, if $K \subseteq \{0, 1\}^n$ is the domain where the limited membership query answers correctly then the elements in \overline{K} (where the limited membership query answers “I don’t know”) are allowed to be classified arbitrarily by the final hypothesis of the learning algorithm.

In this model, the goal of the learner is to run in polynomial time in $n(f)$, $\text{size}(f)$ and some measure of \mathcal{K} and output a “non-strict” correct hypothesis.

Another queries that will be studied are the *malicious membership queries* and *malicious equivalence queries*. In the malicious membership queries MMQ the query can return wrong answers to at most ℓ different assignments for some integer ℓ . In the malicious equivalence query MEQ the query can return a wrong counterexample a (an assignment that is not a counterexample) for at most ℓ different assignments for some integer ℓ . That is, the MEQ can choose any ℓ assignments and can lie on those assignments as many times as it wants. However, if the malicious equivalence query returns a true counterexample a it cannot lie on a afterwards.

The goal of the learner is to run in polynomial time in $n(f)$, $\text{size}(f)$ and ℓ and find a function that equivalent to the target on the points that the queries didn’t lie.

4 Learning using a Complete Set of Terms

In this section we show how to use complete sets for learning. The technique we use here is an extension of the divide and conquer method developed in [B97].

Let \mathcal{T} be a complete set of terms. For each term $T \in \mathcal{T}$ we define the corresponding partial assignment p^T where $p_i^T = 1$ if x_i appears positive in T , $p_i^T = 0$ if it appears negative and $p_i^T = x_i$ otherwise. The partial assignment p^T represents the set of all assignments that satisfy the term T .

Let C be a concept class that is closed under projection and let \mathcal{A} be a learning algorithm that learns C from membership and equivalence queries.

Notice that

Lemma 7 *For every boolean function f and every complete set of terms \mathcal{T} we have*

$$f(x) = \left(\bigvee_{T \in \mathcal{T}} T(x) \cdot f_{p^T}(x) \right),$$

where $f_{p^T} = f(p^T) \in C$.

Proof. Let a be any assignment. By the definition of complete set of terms there is exactly one term $T_0 \in \mathcal{T}$ such that $T_0(a) = 1$. Since $p^{T_0}(a) = a$ we have

$$f(a) = T_0(a)f_{p^{T_0}}(a) = f(p^{T_0})(a) = f(p^{T_0}(a)) = f(a). \square$$

To learn f in the above representation we run $|\mathcal{T}|$ algorithms, one for each f_{p^T} where $T \in \mathcal{T}$. Denote by \mathcal{A}_T the algorithm that is running for learning f_{p^T} . The membership query $\text{MQ}(a)$ for f_{p^T} can be simulated by asking $\text{MQ}(p^T(a))$ for f . This is because $f_{p^T}(a) = f(p^T)(a) = f(p^T(a))$. To simulate equivalence queries we wait until all the algorithms $\mathcal{A}_T, T \in \mathcal{T}$, ask equivalence queries $\text{EQ}(h_T)$ or output h_T and then ask equivalence query $\text{EQ}(H)$ where

$$H(x) = \left(\bigvee_{T \in \mathcal{T}} T(x) \cdot h_T(x) \right).$$

Since \mathcal{T} is complete set of terms, a counterexample b will be a counterexample for the hypothesis $h_{T'}$ for which $T'(b) = 1$. This is because $h_{T'}(b) = H(b) \neq f(b) = f_{p^{T'}}(b)$. We use this counterexample to continue running $\mathcal{A}_{T'}$ until another equivalence query is asked by $\mathcal{A}_{T'}$ or $\mathcal{A}_{T'}$ outputs some hypothesis $h_{T'}$.

Given a class C that is closed under projection and a learning algorithm \mathcal{A} for C . The algorithm **LEARN**($\mathcal{A}, C, \mathcal{T}$) in Figure 1 learns C with a complete set of terms \mathcal{T} . It runs the algorithm \mathcal{A} for every term $T \in \mathcal{T}$ with the changes in steps (3) and (4). In step (4) the algorithm \mathcal{A}_T waits and **LEARN**($\mathcal{A}, C, \mathcal{T}$) runs \mathcal{A}_T for a new term $T \in \mathcal{T}$. When all the algorithms are in the “wait” state, **LEARN**($\mathcal{A}, C, \mathcal{T}$) continues in step (5). It asks equivalence query with

$$H = \left(\bigvee_{T \in \mathcal{T}} T(x) \cdot h_T(x) \right).$$

LEARN($\mathcal{A}, C, \mathcal{T}$)

1. For every $T \in \mathcal{T}$
2. Run $\mathcal{A}_T \equiv \mathcal{A}$ to learn f_{p^T} with the following changes:
3. If \mathcal{A}_T asks MQ(a) then ask MQ($p^T(a)$).
4. If \mathcal{A}_T asks EQ(h_T) or outputs h_T then wait.
5. Define $H = (\bigvee_{T \in \mathcal{T}} T(x) \cdot h_T(x))$. If no \mathcal{A}_T ask EQ then output H else ask EQ(H)
6. If the answer is “Yes” then halt and output(H).
7. If the answer is “No” with b then let $T' \in \mathcal{T}$ such that $T'(b) = 1$.
8. Return b to the algorithm $\mathcal{A}_{T'}$ and continue running $\mathcal{A}_{T'}$ with the following changes:
9. If $\mathcal{A}_{T'}$ asks MQ(a) then ask MQ($p^{T'}(a)$).
10. If $\mathcal{A}_{T'}$ asks EQ($h_{T'}$) or outputs $h_{T'}$ then wait
11. Goto 5.

Figure 1: *Learning given an algorithm \mathcal{A} for C and a complete set of terms \mathcal{T} .*

A counterexample will be a counterexample for one of the algorithms $\mathcal{A}_{T'}$. **LEARN**($\mathcal{A}, C, \mathcal{T}$) continues to run $\mathcal{A}_{T'}$ (steps (8-10)) until it asks another equivalence query or outputs a hypothesis. Then it waits and **LEARN**($\mathcal{A}, C, \mathcal{T}$) again asks an equivalence query.

Proposition 8 *If algorithm \mathcal{A} runs in time t with m membership queries and e equivalence queries then **LEARN**($\mathcal{A}, C, \mathcal{T}$) learns C in time $O(|\mathcal{T}|t)$ with $|\mathcal{T}|m$ membership queries and $|\mathcal{T}|e$ equivalence queries.*

In particular

Proposition 9 *If algorithm \mathcal{A} runs in time t with m membership queries then **LEARN**($\mathcal{A}, C, \mathcal{T}$) learns C in time $O(|\mathcal{T}|t)$ with $|\mathcal{T}|m$ membership queries.*

We use this technique to handle lies in the queries. If a learning algorithm \mathcal{A} for C fails to learn f (because of some lie in one of the queries) one may try to build an appropriate complete set \mathcal{T} and learn f as described in this section. This may not correct the lie but it may isolate it in a smaller domain.

In the next section we will run **LEARN**($\mathcal{A}, C, \mathcal{T}$) with a set of terms \mathcal{T} that are disjoint terms but not complete. In this case the algorithm may receive a counterexample a that satisfies $T(a) = 0$ for all $T \in \mathcal{T}$. We use this counterexample to modify \mathcal{T} and run **LEARN**($\mathcal{A}, C, \mathcal{T}$) again.

5 The Algorithm with Limited Membership Queries

In this section we will study exact learning with equivalence and limited membership queries. For a class \mathcal{K} of subsets of $\{0, 1\}^n$ we will consider the limited membership query $\text{LMQ}_{\mathcal{K}}$. In this

LMQ-LEARN($\mathcal{A}, \mathcal{B}, C, \mathcal{K}$)

1. Run \mathcal{B} with the following changes:
2. If \mathcal{B} asks $\text{MQ}(a)$ then ask $\text{LMQ}_{\mathcal{K}}(a)$.
3. If the answer is “I don’t know” then return 0 else return 1.
4. If \mathcal{B} asks $\text{EQ}(g)$ or outputs g then wait.
5. $\mathcal{T} \leftarrow$ the set of terms of g .
6. Run $W = \text{LEARN}(\mathcal{A}, C, \mathcal{T})$ with $\text{LMQ}_{\mathcal{K}}$ queries instead of MQ .
7. If W asks $\text{LMQ}_{\mathcal{K}}(a)$ and it returns “I don’t know” then Goto 11.
8. If W asks $\text{EQ}(H)$ and it returns a and $g(a) = 0$ then Goto 11.
9. If W outputs H then
10. Ask $\text{EQ}(H)$; If it returns a then Goto 11 else $\text{Output}(H)$.
11. Send a as a counterexample to \mathcal{B} .
12. Goto 1 to Continue running \mathcal{B} .

Figure 2: *Learning with limited membership queries*

model an adversary chooses some $K \in \mathcal{K}$ and then $\text{LMQ}_{\mathcal{K}}(a) = f(a)$ if $a \in K$ and $\text{LMQ}_{\mathcal{K}}(a) =$ “I don’t know” if $a \notin K$. In this model we define a hypothesis to be “non-strict” correct if it agrees with the target concept on all examples except possibly ones for which the limited membership query answers “I don’t know”. Thus, if $K \subseteq \{0, 1\}^n$ is the domain where the limited membership query answers correctly then the elements in \bar{K} (where the limited membership query answers “I don’t know”) are allowed to be classified arbitrarily by the final hypothesis of the learning algorithm. The equivalence query always gives counterexamples from K and it answers “Yes” if the hypothesis is non-strictly correct.

In this section we show that if C is learnable from membership and equivalence queries in time t and \mathcal{K} is learnable as disjoint DNF (the hypotheses in the equivalence queries are disjoint DNF) from membership and equivalence queries in time t' then C is learnable from limited membership queries $\text{LMQ}_{\mathcal{K}}$ and equivalence queries in time $\text{poly}(t, t')$.

One interesting case is when \mathcal{K}_{ℓ} is the set of all sets $\{0, 1\}^n \setminus L$ where $|L| \leq \ell$. In this case $\text{LMQ}_{\mathcal{K}_{\ell}}$ is the limited membership query that answers “I don’t know” on at most ℓ assignments. We denote this query by LMQ_{ℓ} . We will show that \mathcal{K}_{ℓ} is learnable in time $O(n\ell)$ as disjoint DNF from equivalence queries only and therefore we get the result of learning with ℓ “I don’t know” answers to the membership queries.

Let C be a concept class that is closed under projection. Let \mathcal{A} be an algorithm that learns C with membership and equivalence queries. Let \mathcal{B} be the algorithm that learns \mathcal{K} as disjoint DNF with membership and equivalence queries. Consider the algorithm **LMQ-LEARN**($\mathcal{A}, \mathcal{B}, C, \mathcal{K}$) in Figure 2. This algorithm runs algorithm \mathcal{B} first and for each membership query it changes the “I don’t know” answers to 0 and the 0, 1 answers to 1. So \mathcal{B} tries to learn χ_K . When \mathcal{B} asks equivalence query with a disjoint DNF function g it waits and then **LMQ-LEARN**($\mathcal{A}, \mathcal{B}, C, \mathcal{K}$) runs **LEARN**($\mathcal{A}, C, \mathcal{T}$) where \mathcal{T} is the set of terms of g . Notice that \mathcal{T} contains disjoint terms

but may not be a complete set of terms. So, three things may happen that disturb running $\mathbf{LEARN}(\mathcal{A}, C, \mathcal{T})$:

1. The limited membership query $\text{LMQ}_{\mathcal{K}}(a)$ answers “I don’t know”. In this case a is returned as a negative counterexample for the algorithm \mathcal{B} and algorithm \mathcal{B} continues to run.
2. The equivalence query answers an assignment a and all the terms $T \in \mathcal{T}$ satisfy $T(a) = 0$. In this case we know that algorithm $\mathbf{LEARN}(\mathcal{A}, C, \mathcal{T})$ cannot continue running. Then a is returned to \mathcal{B} as a positive counterexample.
3. The algorithm outputs a hypothesis H which is not equivalent to the target. In this case we ask equivalence query and return the answer as a positive counterexample.

Now, we only need to prove that in all the three cases a is indeed a counterexample for g and when \mathcal{B} learns K then $\mathbf{LEARN}(\mathcal{A}, C, \mathcal{T})$ learns a non-strictly correct hypothesis.

We show the following

Theorem 10 *Let C be a class that is closed under projection. Suppose \mathcal{K} is learnable in polynomial time as disjoint DNF of size s from m_1 membership queries and e_1 equivalence queries and C is learnable in polynomial time from m membership queries and e equivalence queries. Then C is learnable in polynomial time with $(e_1 + 1)sm + m_1$ limited membership queries $\text{LMQ}_{\mathcal{K}}$ and $(e_1 + 1)(se + 1)$ equivalence queries.*

Proof of Theorem 10. The correctness of the learning algorithm is obvious since the algorithm stops only when the answer to the equivalence query is “YES”.

We first show that a is indeed a counterexample for g .

Case I. If $\mathbf{LEARN}(\mathcal{A}, C, \mathcal{T})$ asks $\text{LMQ}_{\mathcal{K}}(a)$ then we know that $T(a) = 1$ for some $T \in \mathcal{T}$ (see Figure 1) and therefore $g(a) = 1$. If $\text{LMQ}_{\mathcal{K}}(a)$ answers “I don’t know” then $a \notin K$ and $\chi_{\mathcal{K}}(a) = 0$. Therefore $g(a) \neq \chi_{\mathcal{K}}(a)$ and a is a negative counterexample for g .

Case II. If $\mathbf{LEARN}(\mathcal{A}, C, \mathcal{T})$ asks $\text{EQ}(H)$ and the assignment a that is received by the equivalence query satisfies $g(a) = 0$ then we know that $a \in K$ because the equivalence query does not return assignments from \overline{K} . Therefore $\chi_{\mathcal{K}}(a) = 1 \neq g(a)$ and a is a positive counterexample for g .

Case III. If $\mathbf{LEARN}(\mathcal{A}, C, \mathcal{T})$ outputs

$$H = \left(\bigvee_{T \in \mathcal{T}} T(x) \cdot h_T(x) \right)$$

then we know that all the algorithms \mathcal{A}_T , $T \in \mathcal{T}$ halt and output hypotheses and therefore $h_T(x) = f_{p^T}(x)$ for all $T \in \mathcal{T}$. This implies that $H(x) \Rightarrow f(x)$. Therefore, the counterexample a will satisfy $H(a) = 0$ and $f(a) = 1$. This implies that $g(a) = 0$ and as in the second case $\chi_{\mathcal{K}}(a) = 1 \neq g(a)$ and a is a positive counterexample for g .

Now we show that when \mathcal{B} learns $\chi_{\mathcal{K}}$ then $\mathbf{LEARN}(\mathcal{A}, C, \mathcal{T})$ learns C . If \mathcal{B} learns K then $\chi_{\mathcal{K}} = g = \bigvee_{T \in \mathcal{T}} T$. Then by the above argument $\text{LMQ}_{\mathcal{K}}(a)$ will not answer “I don’t know” and $\text{EQ}(H)$ will not answer an assignment a such that $g(a) = 1$.

Now for the complexity of the algorithm, algorithm **LEARN**($\mathcal{A}, C, \mathcal{T}$) runs at most $e_1 + 1$ times and each time with at most $|\mathcal{T}|m \leq sm$ membership queries and $|\mathcal{T}|e + 1 \leq se + 1$ equivalence queries. Algorithm \mathcal{B} asks m_1 membership queries but the answers for its equivalence queries are received from **LEARN**($\mathcal{A}, C, \mathcal{T}$). This follows the result. \square

In particular we have

Corollary 11 *Let C be a class that is closed under projection. Suppose \mathcal{K} is learnable in polynomial time as disjoint DNF of size s from m_1 membership queries and C is learnable in polynomial time from m membership queries only. Then C is learnable in polynomial time with $sm + m_1$ limited membership queries $LMQ_{\mathcal{K}}$ only.*

When the limited membership query answers “I don’t know” on at most ℓ points then \mathcal{K}_{ℓ} is the class of sets $\{0, 1\}^n \setminus L$ where $|L| \leq \ell$. This class can be learned with ℓ equivalence queries only with hypotheses that are decision trees of size ℓn as follows: We ask EQ(1) and get an assignment in L . After learning q assignments $a_1, \dots, a_q \in L$ we build a decision tree for $h = \chi_{\{a_1, \dots, a_q\}}$ as defined in Lemma 5 and ask equivalence query EQ(h). A new assignment a_{q+1} from L must be returned by the equivalence query. Since by Lemma 5 the size of the decision tree for h is at most ℓn we have the following.

Corollary 12 *Let C be a class that is closed under perfect projection. If C is learnable in time t from m membership queries and e equivalence queries then C is learnable in time $O(n\ell^2 t)$ from $O(n\ell^2 m)$ limited membership queries LMQ_{ℓ} and $O(n\ell^2 e)$ equivalence queries.*

In the next subsection we show that the algorithm **LMQ $_{\ell}$ -LEARN**(\mathcal{A}, t, C) in Figure 4 gives a complexity that linear on ℓ rather than quadratic.

In [B97] Bshouty showed that the class F_k that contains all the functions $f(Q_1, \dots, Q_k)$ where each Q_i is either a term or a clause and f is any boolean function, is learnable as a decision tree of size $s = O\left(\binom{n}{k}\right)$ in $e_1 = O\left(\binom{n}{k}\right)$ equivalence queries and $m_1 = O\left(\binom{n}{k}n\right)$ membership queries. Notice that this class includes the classes k -term DNF and k -clause CNF. By Theorem 10 we have

Corollary 13 *Let C be a class that is closed under projection. If C is learnable in polynomial time from m membership queries and e equivalence queries. Then C is learnable in polynomial time with at most $O(n^{2k}m)$ limited membership queries LMQ_{F_n} (or $LMQ_{\overline{F_n}}$) and $O(n^{2k}e)$ equivalence queries.*

It is also shown in [B97] that the class M_k that contains all the functions $f(P_1, \dots, P_k)$ where P_i are monotone terms and f is any boolean function, is learnable as a decision tree of size $O(n^k)$ from $O(n^{2k})$ membership queries. By Corollary 11 we have

Corollary 14 *Let C be a class that is closed under projection. If C is learnable in polynomial time from m membership queries. Then C is learnable in polynomial time with at most $O(n^{3k}m)$ limited membership queries LMQ_{M_k} (or $LMQ_{\overline{M_k}}$).*

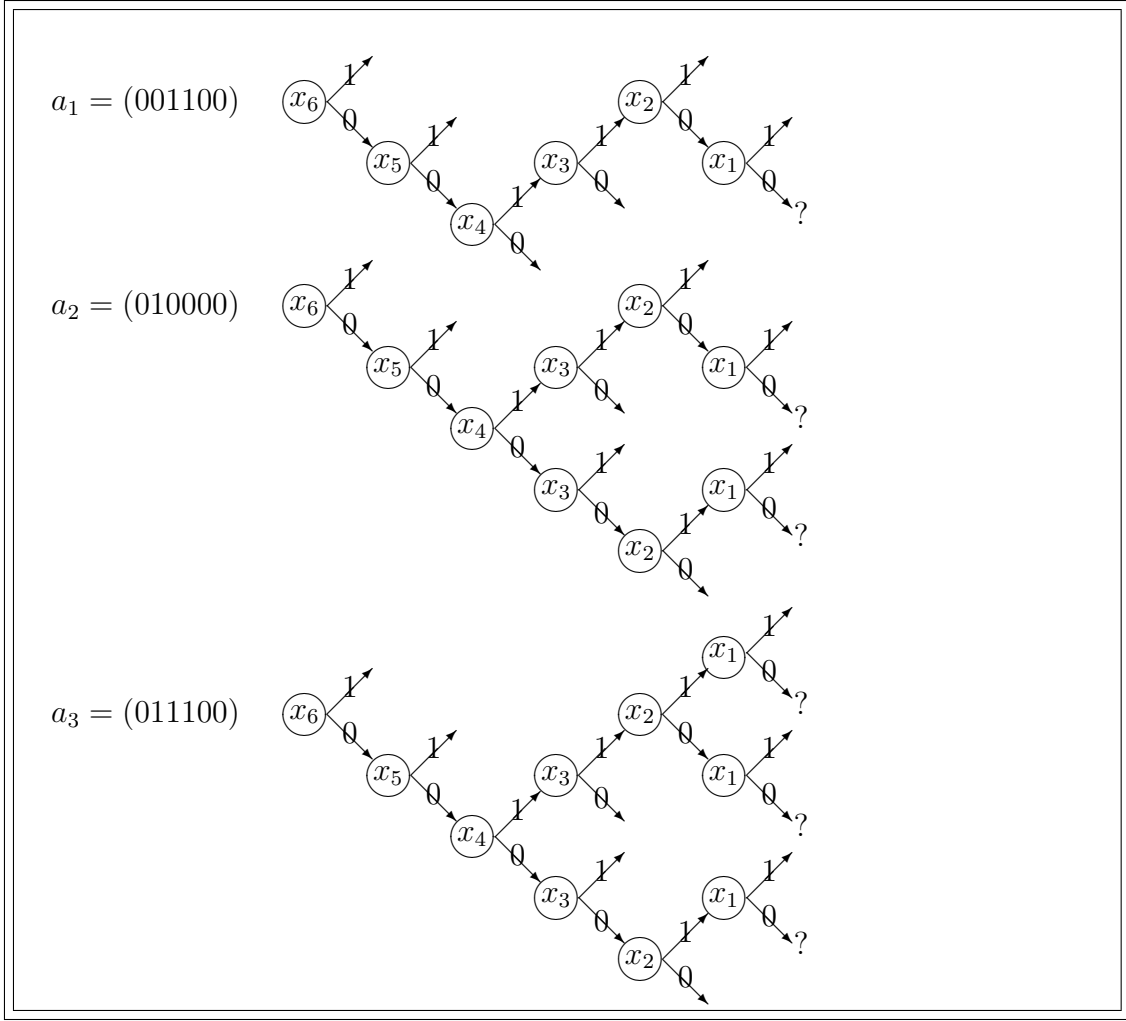


Figure 3: Example for learning with LMQ_ℓ

5.1 The Algorithm with LMQ_ℓ

In this subsection we prove the following

Theorem 15 *Let C be a class that is closed under perfect projection. If C is learnable in time t from m membership queries and e equivalence queries then C is learnable in time $O(nlt)$ from $O(nlm)$ limited membership queries LMQ_ℓ and $O(nle)$ equivalence queries.*

The same algorithm also gives the following:

Corollary 16 *Let C be a class that is closed under perfect projection. If C is learnable in time t from m membership queries only then C is learnable in time $O(nlt)$ from $O(nlm)$ limited membership queries LMQ_ℓ only.*

Let \mathcal{A} be an algorithm that learns C from membership and equivalence queries. We will assume that when the number of variables $n = 0$ (the target then is a constant function) the

LMQ_ℓ-LEARN(\mathcal{A}, C)

1. $\mathcal{T} = \{1\}$; $q_1 \leftarrow 0$; $\mathcal{A}_1 \equiv \mathcal{A}$.
2. While there exists $T \in \mathcal{T}$ with $q_T = 0$.
3. $q_T \leftarrow 1$.
4. Continue to Run \mathcal{A}_T to learn f_{p^T} and at each step do.
5. If \mathcal{A}_T asks MQ(a) then ask LMQ_ℓ($p^T(a)$)
6. If LMQ_ℓ answers “I don’t know” then
7. $d \leftarrow p^T(a)$.
8. $\mathcal{T} \leftarrow (\mathcal{T} \setminus \{T\}) \cup \{T' \in \mathcal{T}^d : |T'| > |T|\}$.
9. For every $T' \in \mathcal{T}^d$ with $|T'| > |T|$ do
10. $q_{T'} = 0$; $\mathcal{A}_{T'} \equiv \mathcal{A}$.
11. Discard algorithm \mathcal{A}_T .
12. If \mathcal{A}_T asks EQ(h_T) or output h_T then wait.
13. Ask EQ(H) where $H = (\bigvee_{T \in \mathcal{T}} T(x) \cdot h_T(x))$.
14. If the answer is “Yes” then halt and output(H).
15. If the answer is b then let $T' \in \mathcal{T}$ such that $T'(b) = 1$
16. Return b to the algorithm $\mathcal{A}_{T'}$.
17. $q_{T'} \leftarrow 0$.
18. Goto 2.

Figure 4: *Learning with LMQ_ℓ.*

algorithm just asks EQ(0) and then EQ(1). One of them will answer “Yes”.

Before we give a formal proof to our main theorem we show how the algorithm runs on a simple example. See Figure 3 and 4.

Suppose we run the algorithm for $n = 6$. The algorithm starts by defining $\mathcal{T} = \{1\}$ and $\mathcal{A}_1 \equiv \mathcal{A}$ and runs \mathcal{A}_1 as long as no membership query answers “I don’t know” (steps 4-5). If \mathcal{A}_1 receives “I don’t know” for $a_1 = (001100)$ then the algorithm builds the complete set of terms

$$\mathcal{T} = \mathcal{T}^{a_1} = \{x_6, \bar{x}_6 x_5, \bar{x}_6 \bar{x}_5 \bar{x}_4, \bar{x}_6 \bar{x}_5 x_4 \bar{x}_3, \bar{x}_6 \bar{x}_5 x_4 x_3 x_2, \bar{x}_6 \bar{x}_5 x_4 x_3 \bar{x}_2 x_1, \bar{x}_6 \bar{x}_5 x_4 x_3 \bar{x}_2 \bar{x}_1\}.$$

This corresponds to the first tree in Figure 3. This complete set of terms isolates the assignment a_1 . We discard \mathcal{A}_1 and replace it with the algorithms $\mathcal{A}_T \equiv \mathcal{A}$ for all $T \in \mathcal{T}$ (steps 6-11). The other parts of the algorithm are simply the algorithm **LEARN**($\mathcal{A}, C, \mathcal{T}$).

Suppose another “I don’t know” is received, say $a_2 = (010000)$. This belongs to the algorithm $\mathcal{A}_{\bar{x}_6 \bar{x}_5 \bar{x}_4}$. We then replace $\bar{x}_6 \bar{x}_5 \bar{x}_4$ with (see step 8)

$$\{\bar{x}_6 \bar{x}_5 \bar{x}_4 x_3, \bar{x}_6 \bar{x}_5 \bar{x}_4 \bar{x}_3 \bar{x}_2, \bar{x}_6 \bar{x}_5 \bar{x}_4 \bar{x}_3 x_2 x_1, \bar{x}_6 \bar{x}_5 \bar{x}_4 \bar{x}_3 x_2 \bar{x}_1\},$$

discard $\mathcal{A}_{\bar{x}_6 \bar{x}_5 \bar{x}_4}$ and run new algorithms \mathcal{A}_T for the the new terms. The latter terms isolate the assignment a_2 . The terms in \mathcal{T} after this change correspond to the second tree in Figure 3. The

leaves that are labeled with ? correspond to the assignments a_1 and a_2 . The third tree results from receiving “I don’t know” for the assignment $a_3 = (011100)$.

We now prove our main Theorem

Proof of Theorem 15 Notice that the algorithm is identical to $\text{LEARN}(\mathcal{A}, C, \mathcal{T})$ except that after each “I don’t know” \mathcal{T} is updated. A term $T \in \mathcal{T}$ is removed and its algorithm \mathcal{A}_T is discarded and is replaced with new terms. Therefore it is enough to show the following:

1. At each stage of the algorithm, \mathcal{T} is perfect complete set of terms that isolates all the assignments for which LMQ_ℓ answered “I don’t know”.
2. If LMQ_ℓ answered “I don’t know” for some assignment d then no algorithm \mathcal{A}_T will ask $\text{LMQ}_\ell(d)$ again.

The latter implies that \mathcal{T} is updated at most ℓ times. Since each update discards one algorithm and adds at most n new algorithm \mathcal{A}_T , the number of algorithms that run is at most $n\ell$. This implies the result.

We now prove (1). Suppose \mathcal{T} is perfect complete set of terms and let $T \in \mathcal{T}$. Let d be any assignment such that $T(d) = 1$ and consider

$$\mathcal{T}' = (\mathcal{T} \setminus \{T\}) \cup \{T' \in \mathcal{T}^d : |T'| > |T|\}.$$

Since $T(d) = 1$ we have $\{T' \in \mathcal{T}^d : |T'| > |T|\} = \{TT'' : T'' \in \mathcal{T}^{(d_1, \dots, d_{n-|T|})}\}$ and

$$\mathcal{T}' = (\mathcal{T} \setminus \{T\}) \cup \{TT'' : T'' \in \mathcal{T}^{(d_1, \dots, d_{n-|T|})}\}$$

Now take any assignment a . Since \mathcal{T} is complete set of terms there is $T_1 \in \mathcal{T}$ such that $T_1(a) = 1$ and for every $T_2 \in \mathcal{T} \setminus \{T_1\}$ we have $T_2(a) = 0$. If $T(a) = 0$ then exactly one term in \mathcal{T}' satisfies a . If $T(a) = 1$ then all the terms in $\mathcal{T} \setminus \{T\}$ do not satisfy a and since $\mathcal{T}^{(d_1, \dots, d_{n-|T|})}$ is a complete set of terms exactly one of the terms in $\{TT'' : T'' \in \mathcal{T}^{(d_1, \dots, d_{n-|T|})}\}$ satisfies a . The set \mathcal{T}' is perfect since all the terms in \mathcal{T}' are of the form $T_{a,k}$ for some assignment a and integer k . This proves (1).

To prove (2) notice that the only term that satisfies d is $T_{d,0}$. For this term the algorithm is $EQ(0)$ followed by $EQ(1)$. So no other algorithm will ask the query $\text{LMQ}_\ell(d)$. \square

6 The Algorithm with Malicious Membership Queries

In this section we study learning with malicious membership queries and malicious equivalence queries. In the malicious membership queries MMQ the query can return wrong answers to at most ℓ_1 different assignments. In the malicious equivalence query MEQ the query can return a wrong counterexample a (an assignment that is not a counterexample) for at most ℓ_2 different assignments. That is, the MEQ can choose any ℓ_2 assignments and can lie on those assignments as many times as it wants. If the malicious equivalence query returns a true counterexample a it cannot lie on a afterwards. If we do not have this constraint then learning is impossible because the MEQ can keep on returning the same assignment for any hypothesis. We will use $\ell = \ell_1 + \ell_2$ for the total number of possible lies.

Our algorithm will learn a hypothesis h that is equivalent to the target hypothesis except for the assignments that the MEQ lies on. In that case the MEQ has to eventually return the true values of one of these assignments. The algorithm updates the value of h on this assignment and asks again MEQ.

In this section we prove the following

Theorem 17 *Let C be a class that is closed under perfect projection. If C is learnable in time t from membership and equivalence queries then C is learnable in time $O((\ell+t)^2n)$ from malicious membership queries and malicious equivalence queries.*

In particular,

Corollary 18 *If C is closed under perfect projection and is learnable in polynomial time from membership and equivalence queries then C is learnable in polynomial time from malicious membership queries and malicious equivalence queries.*

In [BBK05] they show that for certain classes a better time complexity can be achieved.

Let \mathcal{A} be a learning algorithm that learns C in time t from membership and equivalence queries. We will assume that the learner knows t . Later in this section we will remove this constraint. We will also assume that the first three commands in algorithm \mathcal{A} are EQ(0); EQ(1); EQ(0);. This just shortens the code of the algorithm.

The learning algorithm **MMQ-LEARN**(\mathcal{A}, t, C) in Figure 5 runs $\mathcal{A}_1 \equiv \mathcal{A}$ that corresponds to the term 1, i.e., it runs **LEARN**($\mathcal{A}, C, \mathcal{T}$) for $\mathcal{T} = \{1\}$ (See steps 2,4,11-18 in the algorithm). If \mathcal{A}_1 runs more than t steps ($t_T > t$ in step 6) or cannot continue running (gets stuck or cannot execute a command) then the learning algorithm knows that the MMQ or MEQ gave at least one wrong answer. The learning algorithm then splits the term 1 into two terms x_n and \bar{x}_n (step 7) and learns the class with the perfect complete set $\mathcal{T} = \{x_n, \bar{x}_n\}$. The algorithm at any stage is simply running **LEARN**($\mathcal{A}, C, \mathcal{T}$) and each time \mathcal{A}_T , for some $T \in \mathcal{T}$, fails to learn, the learning algorithm splits T into two terms $T \cdot x_{n-|T|}$ and $T \cdot \bar{x}_{n-|T|}$.

We will first prove the following claims

Claim 19 *At each step of the learning algorithm **MMQ-LEARN**(\mathcal{A}, t, C), \mathcal{T} is a perfect complete set of terms.*

Proof. From steps 7 and 8 it is clear that $T = x_n x_{n-1} \cdots x_{n-|T|+1}$. If \mathcal{T} is a perfect complete set of terms then since $x_{n-|T|}$ is a variable that is not in T we have that $\mathcal{T} \leftarrow (\mathcal{T} \setminus \{T\}) \cup \{T \cdot x_{n-|T|}, T \cdot \bar{x}_{n-|T|}\}$ is perfect complete. \square

Claim 20 *If $T \in \mathcal{T}$ is a full term, i.e., $T = T_{a,0}$ for some $a \in \{0,1\}^n$, then the learning algorithm stops splitting T .*

Proof. We remind the reader that the first three commands of \mathcal{A} are EQ(0); EQ(1); EQ(0); and if the malicious equivalence query returns a true counterexample a it cannot lie on a afterwards.

When $T = T_{a,0}$ is a full term then $f_{p^T}(x) \equiv f(a)$. The algorithm \mathcal{A}_T first asks EQ(0). If \mathcal{A}_T receives a counterexample b then $T(b) = 1$ and therefore $b = a$. So the only counterexample that \mathcal{A}_T can receive is a . If this is a true counterexample then $f_{p^T}(x) \equiv 1$ and in the second

MMQ-LEARN(\mathcal{A}, t, C)

1. $\mathcal{T} = \{1\}$. (* An initial complete set of terms *)
 $q_1 \leftarrow 0$. (* $q_T = 0$ if \mathcal{A}_T is ready to run*)
 $t_1 \leftarrow 0$. (* t_T is the number of steps executed in \mathcal{A}_T *)
 $\mathcal{A}_1 \equiv \mathcal{A}$. (* The initial algorithm *)
2. While there exists $T \in \mathcal{T}$ with $q_T = 0$
3. $q_T \leftarrow 1$
4. Continue to Run \mathcal{A}_T to learn f_{p^T} and at each step do.
5. $t_T \leftarrow t_T + 1$
6. If $t_T > t$ or \mathcal{A}_T cannot continue running then
 (* $t_T > t$ means that the algorithm runs more than it should *)
7. $n_T \leftarrow n - |T|$; $T_1 \leftarrow T \cdot x_{n_T}$; $T_0 \leftarrow T \cdot \overline{x_{n_T}}$;
 (* Split the term to two terms *)
8. $\mathcal{T} \leftarrow (\mathcal{T} \setminus \{T\}) \cup \{T_0, T_1\}$.
9. $t_{T_0} \leftarrow 0$; $t_{T_1} \leftarrow 0$. $q_{T_0} \leftarrow 0$; $q_{T_1} \leftarrow 0$.
 (* Create two new algorithms, one for each term *)
10. Halt \mathcal{A}_T and create two algorithms $\mathcal{A}_{T_0} \equiv \mathcal{A}$; $\mathcal{A}_{T_1} \equiv \mathcal{A}$.
11. If \mathcal{A}_T asks MQ(a) then ask MMQ($p^T(a)$)
12. If \mathcal{A}_T asks EQ(h_T) or output h_T then wait
13. Ask MEQ(H) where $H = (\bigvee_{T \in \mathcal{T}} T(x) \cdot h_T(x))$.
14. If the answer is “Yes” then halt and output(H).
15. If the answer is b then let $T' \in \mathcal{T}$ such that $T'(b) = 1$
16. Return b to the algorithm $\mathcal{A}_{T'}$. (* b is a counterexample for $\mathcal{A}_{T'}$ *)
17. $q_{T'} \leftarrow 0$. (* $\mathcal{A}_{T'}$ is ready to continue running *)
18. Goto 2

Figure 5: *Learning with malicious membership queries and malicious equivalence queries.*

command EQ(1) the algorithm will not receive again a . If a is not a true counterexample then $f_{p^T}(x) = 0$ and then when \mathcal{A}_T asks EQ(1) in its second command it can only receive the true counterexample a again. Then in the third command EQ(0), \mathcal{A}_T will not receive a again. \square

Let D_T be the decision tree that corresponds to the perfect complete set \mathcal{T} . The i th level of the tree D_T contains all the nodes at depth i (where the root is at level 1).

We now show

Claim 21 *At each level of the tree there is at most ℓ internal nodes.*

Proof. A node is internal if it is split into two nodes. A split happens only when a lie occurs. Since there is at most ℓ lies and each lie belongs to at most one node in level i in the tree D_T , the number of nodes that split into two nodes at level i is at most ℓ . \square

It follows from claim 21 that

Claim 22 *The number of nodes in \mathcal{T} is at most $2n\ell - 2\ell\lfloor\log\ell\rfloor + 2\ell - 1$.*

Proof. Since at level i of D_T there is at most $\min(2^{i-1}, \ell)$ internal nodes the number of internal nodes of D_T is at most $n\ell - \ell\lfloor\log\ell\rfloor + \ell - 1$. Now the result follows because the number of leaves in D_T is the number of nodes plus 1. \square

Claim 23 *If C is learnable from membership and equivalence queries in time t and t is known to the learning algorithm then C is learnable in time at most $cn\ell t$ with malicious membership queries and malicious equivalence queries for some constant c .*

Proof. Since the number of times \mathcal{A} runs is the number of nodes in D_T and the number of nodes in D_T is at most $2n\ell$ the result follows. \square

Notice that the constant c depends on the number of commands in $\mathbf{MMQ-LEARN}(\mathcal{A}, t, C)$ and is known. Now we are ready to prove Theorem 17

Proof of Theorem 17 Since we do not know t and ℓ we use the doubling technique on t and ℓ . That is we run the learning algorithm $\mathbf{MMQ-LEARN}(\mathcal{A}, t, C)$ for $(t, \ell) = (1, 1)$ and if the algorithm does not output a hypothesis after cn steps we double them and run the learning algorithm with $(t, \ell) = (2, 2)$, then $(t, \ell) = (2^2, 2^2)$ etc. The algorithm will halt at stage k when $2^k \geq \max(\ell, t) > 2^{k-1}$. Then the complexity is $cn + c4n + \dots + c2^{2k}n = O((\ell + t)^2n)$. This implies the result in Theorem 17. \square

References

- [AKST97] D. Angluin, M. Krikis, R. H. Sloan, G. Turán. Malicious omissions and errors in answering to membership queries. *Machine Learning*, 28, 211-255 (1997).
- [AS94] D. Angluin, D. K. Slonim. Randomly Fallible Teachers: Learning Monotone DNF with an Incomplete Membership Oracle. *Machine Learning* 14(1): 7-26 (1994).
- [BBK05] R. Bennet, N. H. Bshouty, L. Khoury. A more efficient learning with Limited and Malicious Queries.

- [B97] N. H. Bshouty. Simple Learning Algorithms Using Divide and Conquer. *Computational Complexity* 6(2), 174-194 (1997).
- [BO01] N. H. Bshouty, A. Owshanko. Learning Regular Sets with an Incomplete Membership Oracle. COLT/EuroCOLT 2001: 574-588.
- [BE02] N. H. Bshouty, N. Eiron. Learning Monotone DNF from a Teacher that Almost Does Not Answer Membership Queries. *Journal of Machine Learning Research* 3: 49-57 (2002).
- [BF02] N. H. Bshouty, V. Feldman. On Using Extended Statistical Queries to Avoid Membership Queries. *Journal of Machine Learning Research* 2: 359-395 (2002).
- [RR95] D. Ron, R. Rubinfeld. Learning Fallible Deterministic Finite Automata. *Machine Learning* 18(2-3): 149-185 (1995).
- [FGMP96] M. Frazier, S. A. Goldman, N. Mishra, L. Pitt. Learning from a Consistently Ignorant Teacher. *J. Comput. Syst. Sci.* 52(3): 471-492 (1996).
- [GKS93] S. A. Goldman, M. J. Kearns, R. E. Schapire. Exact Identification of Read-Once Formulas Using Fixed Points of Amplification Functions. *SIAM J. Comput.* 22(4): 705-726 (1993).
- [GM92] S. A. Goldman, H. D. Mathias. Learning k -Term DNF Formulas with an Incomplete Membership Oracle. COLT 1992, 77-84.
- [JSS99] J. Jackson, E. Shamir, C. Shwartzman. Learning with Queries Corrupted by Classification Noise. *Discrete Applied Mathematics* 92(2-3): 157-175 (1999)
- [V84] L. G. Valiant: A Theory of the Learnable. *Commun. ACM* 27(11): 1134-1142 (1984)

7 Appendix

Proof of Lemma 6 Consider the set S of all assignments of weight (number of ones) k for some constant k . Let $T_1 \vee T_1 \vee \dots \vee T_t$ be any DNF for $\overline{\chi_S}$. Consider an assignment b of weight $k + 1$. Since $\overline{\chi_S}(b) = 1$ there is a term T_r that satisfies b . If there is another assignment $b' \neq b$ of weight $k + 1$ that is satisfied by T_r then all the assignment b'' between b and $b \wedge b'$, i.e. $b \geq b'' \geq b \wedge b'$, is satisfied by T_r . Since the weight of $b \wedge b'$ is at most k this implies that some assignment of weight k is satisfied by T_r and therefore is satisfied by $\overline{\chi_S}$. This is a contradiction. Therefore, each assignment b of weight $k + 1$ corresponds to a different term in the DNF of $\overline{\chi_S}$. This implies that

$$\text{size}_{DNF}(\overline{\chi_S}) \geq \binom{n}{k+1}.$$

Therefore, any complete set of terms for S is of size at least

$$\binom{n}{k+1} = \frac{n-k}{k+1} \binom{n}{k} = O(n|S|). \square$$

Before we prove Lemma 4 we give some definitions and preliminary results.

If D is a decision tree that represents the boolean function f then we say that D is a *decision tree for f* . We denote the size of a decision tree (the number of leaves) D by $\text{size}(D)$. The *decision tree size* $\text{size}_{DT}(f)$ of a boolean function f is the minimal size of a decision tree D for f . We now show

Lemma 24 *For any two boolean functions $f : \{0, 1\}^{n_1} \rightarrow \{0, 1\}$ and $g : \{0, 1\}^{n_2} \rightarrow \{0, 1\}$ and two sets of disjoint variables $x = (x_1, \dots, x_{n_1})$ and $y = (y_1, \dots, y_{n_2})$ we have*

$$\text{size}_{DT}(f(x) \oplus g(y)) = \text{size}_{DT}(f(x)) \cdot \text{size}_{DT}(g(y)).$$

Proof. First notice that for any boolean function $h(z)$ and any z_i we have

$$\text{size}_{DT}(h|_{z_i \leftarrow 0}) + \text{size}_{DT}(h|_{z_i \leftarrow 1}) \geq \text{size}_{DT}(h).$$

This is because the left hand size formula is the minimal decision tree size of a decision tree for h providing that z_i is in the root of the tree. If there is a minimal size tree for h that contains z_i in its root then

$$\text{size}_{DT}(h|_{z_i \leftarrow 0}) + \text{size}_{DT}(h|_{z_i \leftarrow 1}) = \text{size}_{DT}(h).$$

We first show

$$\text{size}_{DT}(f(x) \oplus g(y)) \leq \text{size}_{DT}(f(x)) \cdot \text{size}_{DT}(g(y)). \quad (2)$$

Let D_1 and D_2 be minimal size decision trees for $f(x)$ and $g(y)$, respectively. We build a new decision tree D for $f(x) \oplus g(y)$ as follows: Take the decision tree D_1 and replace each leaf in D_1 labeled with ξ with a decision tree $D_2 + \xi$. Here $D_2 + 0 = D_2$ and $D_2 + 1$ is D_2 with opposite labeled leaves, i.e., change each leaf labeled with 0 with a leaf labeled with 1 and vice versa. It

is easy to see that D is a decision tree for $f(x) \oplus g(y)$ and the size of D is exactly the size of D_1 times the size of D_2 . This implies 2.

We now prove

$$\text{size}_{DT}(f(x) \oplus g(y)) \geq \text{size}_{DT}(f(x)) \cdot \text{size}_{DT}(g(y)).$$

We prove the result by induction on $n = n_1 + n_2$. For $n = 3$ we have $(n_1, n_2) = (1, 2)$ or $(2, 1)$ and one of the functions (the one with $n_i = 1$) is the constant function 0 or 1. This case is clear. Let D be a minimal size tree for $f(x) \oplus g(y)$ and let x_i be the variable in the root of D . Then by the induction hypothesis we have

$$\begin{aligned} \text{size}_{DT}(f(x) \oplus g(y)) &= \text{size}_{DT}(f(x)|_{x_i \leftarrow 0} \oplus g(y)) + \text{size}_{DT}(f(x)|_{x_i \leftarrow 1} \oplus g(y)) \\ &\geq \text{size}_{DT}(f(x)|_{x_i \leftarrow 0})\text{size}_{DT}(g(y)) + \text{size}_{DT}(f(x)|_{x_i \leftarrow 1})\text{size}_{DT}(g(y)) \\ &= \text{size}_{DT}(g(y))(\text{size}_{DT}(f(x)|_{x_i \leftarrow 0}) + \text{size}_{DT}(f(x)|_{x_i \leftarrow 1})) \\ &\geq \text{size}_{DT}(g(y))\text{size}_{DT}(f(x)). \square \end{aligned}$$

A *disjoint CDNF* is a pair of Disjoint DNF (P, Q) such that $\bar{P} = Q$. The size of a disjoint CDNF (P, Q) is the total number of terms in P and Q . For a boolean function f the disjoint CDNF size $\text{size}_{DCD}(f)$ of f is the minimal size of a disjoint CDNF that represents f . We now show

Lemma 25 *For any two boolean functions $f : \{0, 1\}^{n_1} \rightarrow \{0, 1\}$ and $g : \{0, 1\}^{n_2} \rightarrow \{0, 1\}$ and two sets of disjoint variables $x = (x_1, \dots, x_{n_1})$ and $y = (y_1, \dots, y_{n_2})$ we have*

$$\text{size}_{DCD}(f(x) \oplus g(y)) \leq \text{size}_{DCD}(f(x)) \cdot \text{size}_{DCD}(g(y)).$$

Proof. Let (P_1, Q_1) and (P_2, Q_2) be a disjoint CDNF for $f(x)$ and $g(y)$ of size s_1 and s_2 , respectively. Then

$$(P_1Q_2 \vee P_2Q_1, P_1P_2 \vee Q_1Q_2)$$

is a disjoint CDNF for $f(x) \oplus g(y)$ of size $s_1 \cdot s_2$. \square

We now prove our main result

Lemma 26 *If there is a function f with $\text{size}_{DCD}(f) \leq \alpha$ and $\text{size}_{DT}(f) \geq \beta$ then for any s there is a boolean function g with $\text{size}_{DCD}(g) \leq s$ and*

$$\text{size}_{DT}(g) \geq s^{\frac{\log \beta}{\log \alpha} - o(1)}.$$

In particular, there is a set of assignments S that has a complete set that isolates it of size s but any complete set that isolates S that correspond to a decision tree is of size at least $s^{\frac{\log \beta}{\log \alpha} - o(1)}$.

Proof. Let t be an integer such that $s/\alpha < \alpha^t \leq s$. Define $g = f(x^{(1)}) \oplus f(x^{(2)}) \oplus \dots \oplus f(x^{(t)})$. By Lemma 25 we have $\text{size}_{DCD}(g) \leq \alpha^t \leq s$ and by Lemma 24 we have $\text{size}_{DT}(g) \geq \beta^t$. Since

$$\begin{aligned} \beta^t &= \alpha^{\frac{\log \beta}{\log \alpha} t} \\ &\geq \left(\frac{s}{\alpha}\right)^{\frac{\log \beta}{\log \alpha}} \\ &= \frac{s^{\frac{\log \beta}{\log \alpha}}}{\beta} = s^{\frac{\log \beta}{\log \alpha} - o(1)} \end{aligned}$$

the result follows. \square

Since $S = \{000, 111\}$ has a complete set of terms that isolates it of size 5 and since any decision tree for χ_S is of size at least 6 we have

Lemma 27 *There is a set of assignments S that has a complete set that isolates it of size s but any complete set that isolates S that corresponds to a decision tree is of size at least*

$$s^{\frac{\log 6}{\log 5} - o(1)} \geq s^{1.11328275}.$$

Proof. We take the set $S = \{(000), (111)\}$. Then

$$(x_1x_2x_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3, x_1\bar{x}_3 \vee \bar{x}_1x_2 \vee \bar{x}_2x_3)$$

is a disjoint CDNF for χ_S . It is easy to see that any decision tree for χ_S has size at least 6. Then the result follows from Lemma 26. \square