Exact Learning Composed Classes with a Small Number of Mistakes

Nader H. Bshouty and Hanna Mazzawi

Department of Computer Science
Technion, Haifa, 32000, Israel
{bshouty, hanna}@cs.technion.ac.il

Abstract. The Composition Lemma is one of the strongest tools for learning complex classes. It shows that if a class is learnable then composing the class with a class of polynomial number of concepts gives a learnable class. In this paper we extend the Composition Lemma as follows: we show that composing an attribute efficient learnable class with a learnable class with polynomial shatter coefficient gives a learnable class.

This result extends many results in the literature and gives polynomial learning algorithms for new classes.

1 Introduction

The Composition Lemma is one of the strongest tools for learning complex classes. It shows that if a class C is learnable then composing C with polynomial number of concepts G gives a learnable class $C(G)$. This Lemma is used for learning k-CNF of size s (CNF with s terms where each term is of size at most k) in time $O(n^k)$ and $O(sk\log n)$ equivalence queries, k-DL (decision list with terms in the nodes of size at most k) in time $O(n^{3k})$ and $O(n^{2k})$ equivalence queries. Those results was later applied to learning decision tree and CDNF (boolean functions with polynomial size CNF and DNF) in quasi-polynomial time [6, 3], and DNF in sub-exponential time [4, 8, 13].

In this paper we extend the Composition Lemma as follows: We show that composing an attribute efficient learnable class with a learnable class with polynomial shatter coefficient gives a learnable class. Since classes of constant VC dimension has polynomial shatter coefficient, we can apply our result for any class of constant VC dimension.

The following subsections give some results and compare them with the results known from the literature.

1.1 Conjunction of Concepts

Let C be a class with constant VC-dimension, d, that is learnable in polynomial time with q equivalence queries. It is known from [5] that $\bigwedge_k C = \{g_1 \wedge \cdots \wedge g_k \mid g_i \in C\}$ is learnable in $(2kq)^{dd^\perp}$ time and equivalence queries where $d^\perp$ is the VC-dimension of the dual class[1] of C. In most applications $d^\perp \geq d$ (for

[1] The dual class $C^\perp$ of C is the set of functions $g_x : C \to \{0,1\}$ where $g_x(f) = f(x)$.

example halfspaces), which gives at least $(2kq)^{d^2}$ time and query complexity. Our algorithm in this paper runs in time $O((k^2q)^{d+2})$ and asks

$$O(k^2 q \log(kq))$$

equivalence queries. This significantly improves the query complexity of the algorithm.

Our algorithm also runs in polynomial time for classes with polynomial shatter coefficient. This cannot be handled by the previous technique developed in [5]. In particular, if $C_1, C_2, \ldots, C_\ell$ are learnable classes with polynomial shatter coefficient then

$$C_1 \wedge C_2 \wedge \cdots \wedge C_\ell = \{f_1 \wedge f_2 \wedge \cdots \wedge f_\ell \mid f_i \in C_i\}$$

is learnable.

1.2 Halfspace of Functions and Other Classes

Let C be a class with polynomial shatter coefficient that is learnable in polynomial time. Then the class $\mathrm{HS}(C) = \{a_1f_1 + a_2f_2 + \cdots + a_\ell f_\ell \geq 0 \mid f_1, \ldots, f_\ell \in C\}$ and k-CNF$(C) = \{H(f_1, \ldots, f_\ell) \mid H$ is k-CNF $\}$ are learnable in polynomial time. Those classes includes many interesting classes. For example, let $X = \{1, 2, \ldots, n\}^d$ and let C be the set of all halfspaces over X that depends on a constant number of variables. Then $\mathrm{HS}(C)$ is the class of depth two Neural Networks with constant fan-in at the hidden nodes [2]. Also the class k-CNF(C) is interesting because it includes the geometric class of union of $k = O(1)$ n-dimensional polytopes with facets that depends on $j = O(1)$ variables. In particular, it includes the class of union of $O(1)$ boxes in the n-dimensional space. In the constant dimensional space it includes the classes: union of any number of polytopes with constant number of facets, a polytope (with any number of facets) and union of constant number of polytopes. In particular, it contains the union of any number of boxes in the constant dimensional space.

In [2], Auer et. al. already showed that $\mathrm{HS}(C)$ and k-CNF(C) are learnable in polynomial time for C that is halfspaces that depends on a constant number of variables. Our result in this paper shows that $\mathrm{HS}(C)$ and k-CNF(C) are learnable in polynomial time for *any* learnable class C with polynomial shatter coefficient.

Another example is the class of boolean functions on strings that is a threshold of weighted substrings. For $w \in \{0,1\}^{\leq n}$ let $f_w : \{0,1\}^n \to \{0,1\}$ be the function $f_w(x) = 1$ if and only if w is a substring of x, i.e., there is i such that $w = x_i x_{i+1} \cdots x_{i+|w|-1}$. Consider the class C of all f_w where w is a string over $\{0,1\}$. Then the class of threshold of weighted substrings is $\mathrm{HS}(C)$. We show that C has a polynomial shatter coefficient and therefore the class $\mathrm{HS}(C)$ is learnable in polynomial time.

2 Preliminaries

Let X be a set of instances. We call X the *instance space.* A *concept over* X is a boolean function $f : X \to \{0,1\}$. A *concept class* C *over* X is a set of concepts over X.

We say that f is *positive* (respectively, *negative*) on $x \in X$ if $f(x) = 1$ (respectively, $f(x) = 0$). We say that f is *positive* (respectively, *negative*) on $X' \subset X$ if for every $x \in X'$ we have $f(x) = 1$ (respectively, $f(x) = 0$).

For a concept f over X and $X' \subseteq X$ we define the *projection* $f|_{X'} : X' \to \{0, 1\}$ where $f|_{X'}(x) = f(x)$ for every $x \in X'$. For a concept class C over X, the *projection of* C over X' is $C|_{X'} = \{f|_{X'} \ : \ f \in C\}$. We say that f_1 *agrees with* f_2 on X' if $f_1|_{X'} \equiv f_2|_{X'}$, i.e., for every $x \in X'$ we have $f_1(x) = f_2(x)$.

Let $Q \subseteq X$. We say that the concept class F over X is *complete* (concept class) for Q (with respect to C) if $C|_Q \subseteq F|_Q$. In other words, for every $f \in C$ there is $h \in F$ that agrees with f on Q. We say that Q is *shattered* by C if $C|_Q = 2^Q$. The Vapnik-Chervonenkis VC-dimension of C, $\mathrm{VCdim}(C)$, is the size of the largest set shattered by C. We define the *shatter coefficient* $\mathcal{S}(C, m)$ to be the maximal size of $C|_Q$ where $|Q| = m$.

The following is proved by Sauer, [12], and independently by Perles and Shelah.

Lemma 1. *Let C be a concept class over X. For a finite set $Q \subseteq X$ and $C' = C|_Q$ we have*

$$|C'| \le g(|Q|, d) \stackrel{\Delta}{=} \sum_{i=0}^{d} \binom{|Q|}{i} \le \left(\frac{e|Q|}{d}\right)^d$$

where $d = \mathrm{VCdim}(C)$.

In particular, there is a complete concept class for Q with respect to C of size at most $g(|Q|, d)$.

Sauer Lemma does not always give the best bound. Consider the following example

Example. Let $X = \Re^n$ where $\Re$ is the set of the real numbers and suppose $n = 2^\ell$. Consider the function $f_{i,a} : X \to \{0, 1\}$ where $f_{i,a}(x_1, \ldots, x_n) = 1$ if and only if $x_i > a$. Consider the concept class $C = \{f_{i,a} \mid i = 1, \ldots, n \text{ and } a \in \Re\}$. The VC-dimension of C is at least $\ell = \log n$ because the set $\{q_1, \ldots, q_\ell\}$ where $\{(q_{1,i}, q_{2,i}, \ldots, q_{\ell,i}) \mid i = 1, \ldots, \ell\} = \{0, 1\}^\ell$ is shattered by the set of functions $\{f_{i,0} \mid i = 1, \ldots, n\} \subset C$. Now Sauer bound for $C|_Q$ gives at least $(|Q|/\log n)^{\log n}$ where it is easy to see that the size of $C|_Q$ is at most $n(|Q| + 1)$.

Therefore, we will sometimes use the following properties to find upper bounds on the shatter coefficient

Lemma 2. *Let C, C_1 and C_2 be concept classes over X and $\sigma : \{0, 1\}^2 \to \{0, 1\}$. Define $\sigma(C_1, C_2) = \{\sigma(f_1, f_2) | \ f_1 \in C_1, f_2 \in C_2\}$ and $C_1 \otimes_\sigma C_2 = \{f : X \times X \to \{0, 1\};\ f(x, y) = \sigma(f_1(x), f_2(y)) \mid f_1 \in C_1, f_2 \in C_2\}$. We have*

1. *$\mathcal{S}(C, m) \le g(m, \mathrm{VCdim}(C))$.*
2. *$\mathcal{S}(C_1 \cup C_2, m) \le \mathcal{S}(C_1, m) + \mathcal{S}(C_2, m)$.*
3. *$\mathcal{S}(\sigma(C_1, C_2), m) \le \mathcal{S}(C_1, m)\mathcal{S}(C_2, m)$.*
4. *$\mathcal{S}(C, m_1 + m_2) \le \mathcal{S}(C, m_1)\mathcal{S}(C, m_2)$.*
5. *$\mathcal{S}(C_1 \otimes_\sigma C_2, m) \le \mathcal{S}(C_1, m)\mathcal{S}(C_2, m)$.*

We will also consider a family of concept classes $\mathcal{C} = \{C^{(n)}\}$, $n = 1, 2, \cdots$ where $C^{(n)}$ is a concept class over an instance space $X^{(n)}$. When it is clear from the context, we will just call $\mathcal{C}$ a concept class.

For the instance space $X^{(n)} = \{0,1\}^n$ and a concept class $\mathcal{C}$, we say that $\mathcal{C}$ is *closed under combinations* (with repetition) if for every $x_{i_1}, \ldots, x_{i_k} \in \{x_1, \ldots, x_\ell\}$ and a concept $\hat{f} \in C^{(k)}$ there is a concept $f \in C^{(\ell)}$ such that

$$\hat{f}(x_{i_1}, \ldots, x_{i_k}) \equiv f(x_1, \ldots, x_\ell).$$

Let $\mathcal{C}_1 = \{C_1^{(n)}\}$ be a concept class over $\{0,1\}^n$ and let C_2 be a concept class over X. The *composition* of the two concept classes is a concept class over X defined as

$$\mathcal{C}_1(C_2) = \{f(p_1, \ldots, p_k) \mid f \in C_1^{(k)}, p_1, \ldots, p_k \in C_2, k = 1, 2, \cdots\}.$$

2.1 The Learning Model

In the *exact learning* model, [1, 9], a *teacher* has a boolean function f, called the *target function*, which is a member of a concept class C over an instance space X. The goal of the *learner* is to find a hypothesis h that is logically equivalent to f. The learner can ask the teacher *equivalence queries*. In each equivalence query the learner sends the teacher a *hypothesis* $h : X \to \{0,1\}$ from some *class of hypothesis* H. The teacher answers "YES" if h is logically equivalent to f, and provides a *counterexample*, x_0 such that $f(x_0) \neq h(x_0)$, otherwise. We will regard the learner as a *learning algorithm*, the teacher as an oracle EQ, and the equivalence query as a call to this oracle, EQ(h).

We say that a learning algorithm $\mathcal{A}$ *learns* C from H in time $t(\mathcal{A})$ and $q(\mathcal{A})$ equivalence queries if for every target function $f \in C$, $\mathcal{A}$ runs in time at most $t(\mathcal{A})$, asks at most $q(\mathcal{A})$ equivalence queries with hypothesis from H and output a hypothesis from H that is logically equivalent to the target function f. If such algorithm exists, then we say that C is *learnable* from H in time $t(\mathcal{A})$ and $q(\mathcal{A})$ equivalence queries.

Throughout the paper we will assume that H is decidable in polynomial time. That is, given a boolean formula h, the learner can decide in polynomial time whether $h \in H$.

For a concept class $\mathcal{C} = \{C^{(n)}\}$ over $\{0,1\}^n$. We say that algorithm $\mathcal{A}$ learns $\mathcal{C}$ from $\mathcal{H} = \{H^{(n)}\}$ in time $t(\mathcal{A}(n))$ and $q(\mathcal{A}(n))$ equivalence queries if for every n, $\mathcal{A}(n)$ learns $C^{(n)}$ from $H^{(n)}$ in time $t(\mathcal{A}(n))$ and $q(\mathcal{A}(n))$ equivalence queries.

3 The Composition Lemma

In this section we prove (for completeness) the following well known composition Lemma [7, 11].

Lemma 3. (**Composition Lemma**) *Let $\mathcal{C} = \{C^{(n)}\}$ be a concept class over $\{0,1\}^n$ that is closed under combinations. Suppose C is learnable from $\mathcal{H}$ in time*

$t(\mathcal{A}(n))$ and $q(\mathcal{A}(n))$ equivalence queries. Let $G = \{g_1, \ldots, g_\ell\}$ be a concept class over X where each g_i is computable in time $Com(G)$. Then, the algorithm $\mathcal{A}(G)$ in Figure 1 learns $\mathcal{C}(G)$ from $\mathcal{H}(G)$ in time $O(\ell \cdot q(\mathcal{A}(\ell)) \cdot Com(G) + t(\mathcal{A}(\ell)))$ and asks $q(\mathcal{A}(\ell))$ equivalence queries.

```
Algorithm $\mathcal{A}(G = \{g_1, \ldots, g_\ell\})$.

1. $\ell \leftarrow |G|$;
2. **Run** $\mathcal{A}(\ell)$ with the following changes in each step
3.      **If** $\mathcal{A}(\ell)$ asks EQ($h(x_1, \ldots, x_\ell)$)
4.           **then** Ask EQ($h(g_1, \ldots, g_\ell)$).
5.                **If** the oracle answers "YES" **then** return($h(g_1, \ldots, g_\ell)$)
6.                **If** the oracle answers $q \in X$
7.                     **then** give $(g_1(q), \ldots, g_\ell(q))$ to $\mathcal{A}(\ell)$
8.      **If** $\mathcal{A}(\ell)$ outputs $h$ **then** return($h(g_1, \ldots, g_\ell)$)
```

Fig. 1. An algorithm that learns $\mathcal{C}(G)$

Proof. Let $\hat{f}(g_{i_1}, \ldots, g_{i_k}) \in \mathcal{C}(G)$ be the target function where $\hat{f} \in C^{(k)}$ and $g_{i_1}, \ldots, g_{i_k} \in G$. Since the concept class is close under combinations, there is a function $f \in C^{(\ell)}$ such that $\hat{f}(g_{i_1}, \ldots, g_{i_k}) \equiv f(g_1, \ldots, g_\ell)$.

Now since each counterexample q for $h(g_1, \ldots, g_\ell)$ satisfies

$$h(g_1(q), \ldots, g_\ell(q)) \neq \hat{f}(g_{i_1}(q), \ldots, g_{i_k}(q)) = f(g_1(q), \ldots, g_\ell(q)),$$

the assignment $(g_1(q), \ldots, g_\ell(q))$ is a counterexample for h with respect to the function f. Since $\mathcal{A}$ learns $\mathcal{C}$ from $\mathcal{H}$, it will learn some $h \in H^{(\ell)}$ that is logically equivalent to the function $f \in C^{(\ell)}$. Then

$$\hat{f}(g_{i_1}, \ldots, g_{i_k}) \equiv f(g_1, \ldots, g_\ell) \equiv h(g_1, \ldots, g_\ell).$$

The algorithm runs in time at most $O(\ell \cdot q(\mathcal{A}(\ell)) \cdot Com(G) + t(\mathcal{A}(\ell)))$ and asks $q(\mathcal{A}(\ell))$ equivalence queries. ■

Notice that when $|G|$ is exponentially large then the complexity is exponential. In the next section we show that, with some constraints on $\mathcal{C}$ and G, a modified version of the composition lemma gives an algorithm with small time and query complexity even when G is exponentially large.

4 The Algorithm

In this section we give our main algorithm. The main idea of our algorithm is the following: The learner wants to learn $\mathcal{C}_1(C_2)$ for a large concept class

C_2 using learning algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ for $\mathcal{C}_1$ and C_2, respectively. Since the complexity in the composition lemma (Lemma 3) depends on $|C_2|$, which may be exponentially large, the learner cannot use the composition lemma. Instead it does the following: Let $f(p_1, \ldots, p_k)$ be the target function that the learner is trying to learn. At some stage of the learning process the learner has a set of examples Q. It uses this set with the algorithm $\mathcal{A}_2$ to learn a complete concept for Q, $G = \{g_1, \ldots, g_\ell\} \subset C_2$ with respect to C_2. This set may not contain $p_1, \ldots, p_k$ but for each p_i there is g_{r_i} that is "close" to p_i. By "close" we mean that g_{r_i} is a hypothesis of some equivalence query in $\mathcal{A}_2$ when $\mathcal{A}_2$ runs with the target p_i over the instance space Q. Then the learner assumes that $p_1, \ldots, p_k \in G$ and runs the algorithm $\mathcal{A}_1(G)$. When $\mathcal{A}_1(G)$ runs more than it should or gets stuck then the learner knows that the assumption was wrong. But fortunately, we are able to prove that one of the new counterexamples the learner obtains from running $\mathcal{A}_1(G)$ provides a counterexample for one of the g_{r_i}. The learner then adds all the counterexamples to Q and runs the algorithm again. Eventually, the set G will contain $p_1, \ldots, p_k$ and $\mathcal{A}_1(G)$ will learn the target function.

We show that if $C_2|_Q$ is small and $\mathcal{A}_1$ has small complexity then $\mathcal{C}_1(C_2)$ is learnable.

In subsection 4.1 we show how to build G and then in subsection 4.2 we give our main algorithm followed by a proof of correctness and complexity analysis.

4.1 Find All Consistent Hypotheses

Let C be a concept class and H be a hypothesis class. Let $\mathcal{A}$ be a learning algorithm that learns C from H in time $t(\mathcal{A})$ and $q(\mathcal{A})$ equivalence queries. In this subsection we give an algorithm that, for a set of points $Q = \{q_1, q_2, \ldots, q_\ell\}$, outputs a set of hypothesis $F \subset H$ such that $C|_Q \subseteq F|_Q$. That is, the algorithm generates $F \subseteq H$ that is complete for Q with respect to C.

The first algorithm in this subsection is **Find_Hypothesis**$(\mathcal{A}, P, Q\backslash P)$ in Figure 2. It searches for a hypothesis $h \in H$ that is positive on P and negative on $Q\backslash P$ for some $P \subseteq Q$. For such hypothesis h we say that h is *consistent with* $(P, Q\backslash P)$. The algorithm is very similar to the algorithm in [5].

We now prove

Fact 1. *Let C be a concept class over X. Let $\mathcal{A}$ be a learning algorithm that learns C from H in time $t(\mathcal{A})$ and $q(\mathcal{A})$ equivalence queries. Let Q be a set of ℓ instances from X and $P \subseteq Q$.* **Find_Hypothesis***$(\mathcal{A}, P, Q\backslash P)$ in Figure 2 runs in time $O(t(\mathcal{A}) + \ell \cdot q(\mathcal{A}))$ and outputs $h \in H$ that satisfies the following:*

1. *If there is $f \in C$ that is consistent with $(P, Q\backslash P)$ then h is consistent with $(P, Q\backslash P)$.*
2. *If there in no $f \in H$ that is consistent with $(P, Q\backslash P)$ then h is "NULL".*
3. *If there is $f \in C$ that is consistent with $(P, Q\backslash P)$ and $\mathcal{A}$ halts then $h \equiv f$.*

Proof. The algorithm runs the learning algorithm $\mathcal{A}$ (line 2), counts the number of its steps (lines 1 and 3) and the number of times it asks equivalence queries (lines 1 and 5). If $\mathcal{A}$ runs more than $t(\mathcal{A})$ steps, asks more than $q(\mathcal{A})$ equivalence queries or gets stuck (this also includes the cases where the algorithm asks EQ(h) or outputs h where $h \notin H$), then it returns "NULL" (lines 3,5 and 13). This indicates that there exists no consistent hypothesis in C for $(P, Q\backslash P)$.

For each equivalence query EQ(h) that $\mathcal{A}$ asks, the algorithm returns to $\mathcal{A}$ a counterexample from P or $Q\backslash P$, i.e., some point $q \in P$ where $h(q) = 0$ or $q \in Q\backslash P$ where $h(q) = 1$ (lines 4-8). Obviously, if the algorithm cannot find such point then the hypothesis h is consistent with $(P, Q\backslash P)$ (line 6-7).

```
Algorithm **Find_Hypothesis**($\mathcal{A}, P, Q\backslash P$).

 1. $time \leftarrow 0$; $query \leftarrow 0$;
 2. **Run** $\mathcal{A}$ with the following changes in each step
 3.       $time \leftarrow time + 1$;
 4.       **If** $\mathcal{A}$ asks EQ($h$) where $h \in H$
 5.             **then** $query \leftarrow query + 1$;
 6.                   **If** $h$ consistent with $(P, Q\backslash P)$
 7.                         **then** return($h$);
 8.                         **else** give $\mathcal{A}$ a counterexample from $P$ or $Q\backslash P$.
 9.       **If** $\mathcal{A}$ outputs $h$
10.             **then If** $h$ consistent with $(P, Q\backslash P)$
11.                         **then** return($h$)
12.                         **else** return($NULL$);
13.       **If** $\mathcal{A}$ cannot execute this step **or** $time > t(\mathcal{A})$ **or** $query > q(\mathcal{A})$
14.             **then** return($NULL$);
```

Fig. 2. An algorithm that finds a hypothesis that is consistent with $(P, Q\backslash P)$

Now if there is $f \in C$ that is consistent with $(P, Q\backslash P)$ then either one of the hypothesis $h \in H$ in the equivalence queries is consistent with $(P, Q\backslash P)$ or, since the learning algorithm $\mathcal{A}$ learns C, the algorithm $\mathcal{A}$ halts and outputs $h \in H$ that is equivalent to f. In both cases, the output hypothesis is in H and consistent with $(P, Q\backslash P)$.

If there is no $f \in C$ that is consistent with $(P, Q\backslash P)$ then either $\mathcal{A}$ outputs an $h \in H$ that is consistent with $(P, Q\backslash P)$, gets stuck, goes into an infinite loop or outputs a hypothesis that is not consistent with $(P, Q\backslash P)$. In the latter three cases the algorithm outputs "NULL". ■

The second algorithm, **Find_Complete_Concept** in Figure 3, finds $F \subseteq H$ that is complete for Q with respect to C. It starts with a hypothesis h_0 that is consistent with $(\emptyset, \emptyset)$ (line 1). At stage i in the **For** command (line 2) the set (of hypothesis h in) $\mathcal{F}_{i-1}$ is complete for $Q_{i-1} = \{q_1, \ldots, q_{i-1}\}$ with respect to C. For each hypothesis g that is consistent with $(P, Q_{i-1}\backslash P)$ (line 4) it runs **Find_Hypothesis** to try to find a hypothesis $g_1 \in H$ that is consistent with $(P \cup \{q_i\}, Q_{i-1}\backslash P)$ (line 5) and a hypothesis $g_2 \in H$ that is consistent with

```
Algorithm Find_Complete_Concept($\mathcal{A}, Q = \{q_1, \ldots, q_\ell\}$).

 1. $h_0 \leftarrow$Find_Hypothesis$(\mathcal{A}, \emptyset, \emptyset)$; $\mathcal{F}_0 \leftarrow \{((\emptyset, \emptyset), h_0)\}$;
 2. For $i = 1$ to $\ell$ do
 3.     $\mathcal{F}_i \leftarrow \emptyset$; $Q_{i-1} \leftarrow \{q_1, \ldots, q_{i-1}\}$;
 4.     For all $((P, Q_{i-1} \backslash P), g) \in \mathcal{F}_{i-1}$ do
 5.         $g_1 \leftarrow$Find_Hypothesis$(\mathcal{A}, P \cup \{q_i\}, Q_{i-1} \backslash P)$.
 6.         $g_2 \leftarrow$Find_Hypothesis$(\mathcal{A}, P, (Q_{i-1} \backslash P) \cup \{q_i\})$.
 7.         If $g_1 \neq$"NULL" then $\mathcal{F}_i \leftarrow \mathcal{F}_i \cup \{((P \cup \{q_i\}, Q_{i-1} \backslash P), g_1)\}$
 8.         If $g_2 \neq$"NULL" then $\mathcal{F}_i \leftarrow \mathcal{F}_i \cup \{((P, (Q_{i-1} \backslash P) \cup \{q_i\}), g_2)\}$
 9. $F \leftarrow \{h \mid ((P, Q \backslash P), h) \in \mathcal{F}_\ell\}$
10.output$(F)$.
```

Fig. 3. An algorithm that outputs a complete concept for Q with respect to C

$(P, (Q_{i-1} \backslash P) \cup \{q_i\})$ (line 6). That is, it assumes that q_i is positive and then tries to find a consistent hypothesis $g_1 \in H$ and then assumes that it is negative and again tries to find a consistent hypothesis $g_2 \in H$. If such hypothesis exists then it puts it in $\mathcal{F}_i$ (lines 7 and 8).

We now show

Fact 2. *Let C be a concept class over X. Let $\mathcal{A}$ be a learning algorithm that learns C from H in time $t(\mathcal{A})$ and $q(\mathcal{A})$ equivalence queries. Let Q be a subset of X.* **Find_Complete_Concept** *$(\mathcal{A}, Q)$ runs in time at most*

$$O\left(|Q|(t(\mathcal{A}) + |Q| \cdot q(\mathcal{A})) \cdot \mathcal{S}(H, |Q|)\right)$$

and outputs $F \subseteq H$ that is complete for Q with respect to C of size at most $\mathcal{S}(H, |Q|)$.

Proof. Obviously, $\mathcal{S}(H, i-1) \le \mathcal{S}(H, i)$ and therefore, $|\mathcal{F}_i| \le |\mathcal{F}_{|Q|}| \le \mathcal{S}(H, |Q|)$. Therefore, the algorithm **Find_Complete_Concept**$(\mathcal{A}, Q)$ runs **Find_Hypothesis** at most $2|Q|\mathcal{S}(H, |Q|)$ times. By Fact 1 the result follows. ■

We will further improve the complexity of **Find_Complete_Concept** and prove some new properties of the algorithm that will be used in the sequel.

First, we will assume that in **Find_Hypothesis** when $\mathcal{A}$ asks equivalence query, the algorithm always chooses the counterexample in Q with the smallest index and sends it to $\mathcal{A}$. See the algorithm in Figure 2 line 8. This requirement is not necessary but it simplifies the analysis. Second, if **Find_Hypothesis** stops in step 7, i.e., the hypothesis in the equivalence query h (in step 4) is consistent with $(P, Q \backslash P)$, then the next time we call **Find_Hypothesis** $(\mathcal{A}, P \cup \{q\}, Q \backslash P)$ and **Find_Hypothesis** $(\mathcal{A}, P, (Q \backslash P) \cup \{q\})$ the following facts are true:

1. The hypothesis h is consistent either with $(P \cup \{q\}, Q \backslash P)$ or $(P, (Q \backslash P) \cup \{q\})$ and therefore for one of them the algorithm **Find_Complete_Concept** does not need to call **Find_Hypothesis**.

2. For the other one, **Find_Hypothesis** does not need to start running $\mathcal{A}$ from the beginning. It can just continue running it from step 4, i.e., returns the counterexample q to $\mathcal{A}$ and continue running $\mathcal{A}$ until either a new consistent hypothesis is found or it returns "NULL".

Third, if **Find_Hypothesis** stops in step 11, then the algorithm can stop calling **Find_Hypothesis** for the descendants of $(P, Q\backslash P)$ and add h to F.

We will call this new algorithm **Find_Complete**. Now we have

Fact 3. *Let C be a concept class over X. Let $\mathcal{A}$ be a learning algorithm that learns C from H in time $t(\mathcal{A})$ and $q(\mathcal{A})$ equivalence queries. Let Q be a subset of X.* **Find_Complete**$(\mathcal{A}, Q)$ *runs in time at most*

$$O\left((t(\mathcal{A}) + |Q| \cdot q(\mathcal{A})) \cdot \mathcal{S}(H, |Q|)\right)$$

and outputs $F \subseteq H$ of size at most $\mathcal{S}(H, |Q|)$ that is complete for Q with respect to C.

Proof. This follows from the fact that the algorithm runs only one time for every hypothesis in $\mathcal{F}_{|Q|}$. ■

4.2 The Main Algorithm

In this section we give our main algorithm.

Let $\mathcal{C}_1 = \{C_1^{(n)}\}$ be a concept class over $\{0,1\}^n$ and $\mathcal{A}_1$ be a learning algorithm for $\mathcal{C}_1$. Let C_2 be a concept class over X and $\mathcal{A}_2$ be a learning algorithm for C_2. Consider the algorithm $\mathcal{A}_1(\mathcal{A}_2)$ in Figure 4. At some stage of the algorithm it has some set of examples Q. It generates a set $G \subseteq H$ that is complete for Q with respect to C_2 (line 2). The algorithm then learns $\mathcal{A}_1(G)$ using the composition Lemma (see lines 4-19). If the algorithm fails (see lines 3, 5, 12-18 and 19) then it reruns the algorithm with the examples in Q and all the counterexamples received from $\mathcal{A}_1(G)$ (see steps 11 and 17).

We prove

Theorem 1. *Let $\mathcal{C}_1 = \{C_1^{(n)}\}$ be a concept class over $\{0,1\}^n$ that is closed under combinations and $\mathcal{A}_1$ be a learning algorithm that learns $\mathcal{C}_1$ from $\mathcal{H}_1$ in time $t(\mathcal{A}_1(n))$ and $q(\mathcal{A}_1(n))$ equivalence queries. Let C_2 be a concept class over X and $\mathcal{A}_2$ be a learning algorithm that learns C_2 from H_2 in time $t(\mathcal{A}_2)$ and $q(\mathcal{A}_2)$ equivalence queries. Then $\mathcal{A}_1(\mathcal{A}_2)$ learns $\mathcal{C}_1(C_2)$ from $\mathcal{H}_1(H_2)$ in time*

$$O\left(\mathcal{S}(H_2, \rho_\tau)(t(\mathcal{A}_2) + \rho_\tau \cdot q(\mathcal{A}_2)) + \tau \cdot t(\mathcal{A}_1(\mathcal{S}(H_2, \rho_\tau)))\right)$$

and ρ_τ equivalence queries where $\tau = q(\mathcal{A}_2) \cdot k$, $\rho_0 = 0$,

$$\rho_{i+1} = \rho_i + q(\mathcal{A}_1(\mathcal{S}(H_2, \rho_i))),$$

and the target function is $f(p_1, p_2, \ldots, p_k)$.

```
Algorithm $\mathcal{A}_1(\mathcal{A}_2)$.

 1. $Q \leftarrow \emptyset$; $s \leftarrow 0$;
 2. $G \leftarrow$ **Find_Complete**$(\mathcal{A}_2, Q)$;
 3. $time \leftarrow 0$; $query \leftarrow 0$; $\ell \leftarrow |G|$;
 4. **Run** $\mathcal{A}_1(\ell)$ with the following changes in each step
 5.       $time \leftarrow time + 1$;
 6.       **If** $\mathcal{A}_1(\ell)$ asks EQ$(h(x_1, \ldots, x_\ell))$
 7.             **then** Ask EQ$(h(g_1, \ldots, g_\ell))$ where $G = \{g_1, \ldots, g_\ell\}$.
 8.                   **If** the oracle answers "YES" **then** return$(h(g_1, \ldots, g_\ell))$
 9.                   **If** the oracle answers $q \in X$
10.                         **then** give $(g_1(q), \ldots, g_\ell(q))$ to $\mathcal{A}_1(\ell)$
11.                               $s \leftarrow s + 1; q_s \leftarrow q; Q \leftarrow Q \cup \{q_s\}$;
12.                               $query \leftarrow query + 1$;
13.       **If** $\mathcal{A}_1(\ell)$ outputs $h$
14.             **then** Ask EQ$(h(g_1, \ldots, g_\ell))$ where $G = \{g_1, \ldots, g_\ell\}$.
15.                   **If** the oracle answer "YES" **then** return$(h(g_1, \ldots, g_\ell))$
16.                   **If** the oracle answer $q \in X$
17.                         **then** $s \leftarrow s + 1; q_s \leftarrow q; Q \leftarrow Q \cup \{q_s\}$;
18.                               **goto** 2.
19.       **If** $\mathcal{A}_1(\ell)$ cannot execute this step **or**
            $time > t(\mathcal{A}_1(\ell))$ **or** $query > q(\mathcal{A}_1(\ell)) + 1$
20.             **then goto** 2.;
```

Fig. 4. An algorithm that learns $\mathcal{C}_1(C_2)$

Proof. Let $f(p_1, \ldots, p_k)$ be the target function where $f \in C_1^{(k)}$ and $p_1, \ldots, p_k \in C_2$. At stage i the algorithm has a set of instances Q collected from the equivalence queries. Since $G \leftarrow$**Find_Complete**$(\mathcal{A}_2, Q)$, for every $P \subseteq Q$, if there is a concept in C_2 that is consistent with $(P, Q \backslash P)$ then there is $g \in G$ that is consistent with $(P, Q \backslash P)$. Therefore, for every $j = 1, \ldots, k$ there is $g_{r_j} \in G$ that is consistent with $(P_j, Q \backslash P_j)$ where $P_j = \{q \in Q \mid p_j(q) = 1\}$. That is, $p_j|_Q = g_{r_j}|_Q$. Each g_{r_j} was obtained by running $\mathcal{A}_2$ with $(P_j, Q \backslash P_j)$ in **Find_Hypothesis**$(\mathcal{A}_2, P_j, Q \backslash P_j)$. We denote by $m(\mathcal{A}_2, P_j, Q \backslash P_j)$ the number of equivalence queries that $\mathcal{A}_2$ asks in **Find_Hypothesis**$(\mathcal{A}_2, P_j, Q \backslash P_j)$ before it outputs g_{r_j}. By Fact 1, if $\mathcal{A}_2$ halts then $p_j \equiv g_{r_j}$ and therefore $m(\mathcal{A}_2, P_j, Q \backslash P_j) \leq q(\mathcal{A}_2)$ for every j.

Now we will show that if the algorithm goes to step 2 (from step 18 or 20), i.e., $\mathcal{A}_1$ fails to find the target, the new set Q' which is Q with the new counterexamples from $\mathcal{A}_1(G)$, satisfies $m(\mathcal{A}_2, P'_j, Q' \backslash P'_j) > m(\mathcal{A}_2, P_j, Q \backslash P_j)$ for at least one j where $P'_j = \{q \in Q' \mid p_j(q) = 1\}$. In other words, one of the new points in Q' is a counterexample for one of the hypothesis g_{r_j}. This will show that after at most $q(\mathcal{A}_2) \cdot k$ stages the set G contains $p_1, \ldots, p_k$. When $p_1, \ldots, p_k$ are in G then the algorithm $\mathcal{A}_1(G)$ (steps 4-20) will learn the target.

We will now show that either one of the new points is a counterexample for one of the g_{r_j} or the learner has learned the target. Suppose none of the points in $Q'\backslash Q$ is a counterexample for $g_{r_1},\ldots,g_{r_k}$. That is, for every $j\le k$ and every $q\in Q'\backslash Q$ we have $g_{r_j}(q)=p_j(q)$. Then, for every $q\in Q'\backslash Q$ we have

$$f(p_1(q),\ldots,p_k(q))=f(g_{r_1}(q),\ldots,g_{r_k}(q)).$$

Since the algorithm runs $\mathcal{A}_1(G)$ and each counterexample for the target $f(p_1,\ldots,p_k)$ is also a counterexample for $f(g_{r_1},\ldots,g_{r_k})$, the algorithm $\mathcal{A}_1(G)$ will learn h that is equivalent to $f(g_{r_1},\ldots,g_{r_k})$. Then, when the algorithm asks equivalence queries with $h\equiv f(g_{r_1},\ldots,g_{r_k})$ it either receives a counterexample q and then for this $q\in Q'\backslash Q$ we have $f(p_1(q),\ldots,p_k(q))\neq h(q)=f(g_{r_1}(q),\ldots,g_{r_k}(q))$ which is a contradiction, or, it receives "YES" and then we have $f(g_{r_1},\ldots,g_{r_k})\equiv h\equiv f(p_1,\ldots,p_k)$. This completes the correctness of the algorithm.

We now prove its complexity. Let ρ_i be the size of $|Q|$ at stage i. Then $\rho_0=0$ and at stage $i+1$ we have $|G|=\mathcal{S}(H_2,\rho_i)$ and therefore $\mathcal{A}_1(G)$ generates $q(\mathcal{A}_1(\mathcal{S}(H_2,\rho_i)))$ more counterexamples. Therefore, $\rho_{i+1}=q(\mathcal{A}_1(\mathcal{S}(H_2,\rho_i)))+\rho_i$. Since the algorithm runs at most $\tau=q(\mathcal{A}_2)\cdot k$ stages, the number of equivalence queries in the algorithm is at most ρ_τ.

The time complexity is the time for **Find_Complete** with ρ_τ examples, which is equal to $O\left(\mathcal{S}(H_2,\rho_\tau)(t(\mathcal{A}_2)+\rho_\tau\cdot q(\mathcal{A}_2))\right)$ plus the time for running $\mathcal{A}_1$ at each stage, which is equal to $\sum_{i=1}^{\tau} t(\mathcal{A}_1(\mathcal{S}(H_2,\rho_i)))\le\tau\cdot t(\mathcal{A}_1(\mathcal{S}(H_2,\rho_\tau)))$. ■

In the following section we give some applications of the main Theorem

5 Applications

In this section we first prove

Theorem 2. *Let $\bigwedge$ be the set of monotone conjunctions (monomials) over $V=\{x_1,x_2,\cdots\}$. Let C be a concept class that is learnable from H in time t and q equivalence queries. Suppose $\mathcal{S}(H,m)\le\gamma m^d$ for some d and $\gamma\ge 2$ that are independent of m. Then $\bigwedge_k C=\{g_1\wedge\cdots\wedge g_k\mid g_i\in C\}$ is learnable in time $O(\gamma\rho^d(t+\rho q))$ and*

$$\rho=O(k^2qd\log(kqd\gamma^{1/d}\log\gamma))$$

equivalence queries.

In particular, when H has polynomial size shatter coefficient then $\bigwedge_k C$ is learnable in time $O(\gamma\rho_0^d(t+\rho_0 q))$ and

$$\rho_0=O(k^2q\log(kq\gamma))$$

equivalence queries.

Proof. We use WINNOW1 for learning $\bigwedge$, [9]. For a conjunction over $\{0,1\}^n$ with k relevant variables, WINNOW1 runs in time $O(nk\log n)$ and asks $ck\log n$ equivalent queries for some constants c. By Theorem 1 the number of equivalence queries ρ satisfies $\rho\le\rho_\tau$ where $\tau=qk$, $\rho_0=0$ and

$$\rho_{i+1} = \rho_i + ck\log(\mathcal{S}(H,\rho_i)) \le \rho_i + cdk\log\rho_i + ck\log\gamma.$$

Then

$$\begin{aligned}\rho_\tau &= \sum_{i=0}^{\tau-1}(\rho_{i+1}-\rho_i)\\ &\le \sum_{i=1}^{\tau-1} cdk\log\rho_i + ck\log\gamma\\ &\le cdk\tau\log\rho_\tau + ck\tau\log\gamma.\end{aligned}$$

Now, using Fact 4 below, we have

$$\rho \le \rho_\tau \le 2cdk\tau\log(c^2dk^2\tau^2\log\gamma) + ck\tau\log\gamma = O(k^2qd\log(kqd\gamma^{1/d}\log\gamma)).$$

By Theorem 1 the time complexity follows. ■

Fact 4. *Let $\alpha,\beta > 2$ be constants and $\rho \ge 1$ that satisfies*

$$\rho \le \alpha\log\rho + \beta.$$

Then

$$\rho \le 2\alpha\log(\alpha\beta) + \beta.$$

Proof. Consider the two increasing monotone functions $f(x) = x$ and $g(x) = \alpha\log x + \beta$ for $x \ge 1$. Both functions intersect at one point ρ_0 that satisfies $\rho_0 = \alpha\log\rho_0 + \beta$. For $x > \rho_0$ we have $f(x) > g(x)$ and for $1 < x < \rho_0$ we have $f(x) < g(x)$. Therefore, it is enough to show that for $\rho_1 = 2\alpha\log(\alpha\beta) + \beta$ we have $g(\rho_1) < f(\rho_1)$.

Now since $\alpha,\beta > 2$ we have

$$\begin{aligned}g(\rho_1) &= \alpha\log(2\alpha\log(\alpha\beta) + \beta) + \beta\\ &< \alpha\log(2\alpha\beta\log(\alpha\beta)) + \beta\\ &< \alpha\log((\alpha\beta)^2) + \beta = \rho_1 = f(\rho_1).\end{aligned}$$ ■

Let C be a concept class with constant VC-dimension, d, that is learnable in polynomial time with q equivalence queries. It is known from [5] that $\bigwedge_k C = \{g_1\wedge\cdots\wedge g_k \mid g_i \in C\}$ is learnable in $(2kq)^{dd^\perp}$ time and equivalence queries where $d^\perp$ is the VC-dimension of the dual concept class of C. In most applications $d^\perp \ge d$ (for example halfspaces), which gives at least $(2kq)^{d^2}$ time and equivalence queries. Theorem 2 shows that this concept class is learnable in time $O((k^2q)^{d+2})$ and

$$O(k^2q\log(kq))$$

equivalence queries. This significantly improves the query complexity in [5].

Our algorithm also runs in polynomial time for concept classes with polynomial shatter coefficient. This cannot be handled by the previous technique developed in [5].

We now show

Corollary 1. *Let C_i be a concept class that is learnable from H_i in time t_i and q_i equivalence queries for $i = 1, \dots, k$. Suppose $\mathcal{S}(H_i, m) \le \gamma_i m^{d_i}$ for $i = 1, \dots, k$ where each d_i and $\gamma_i > 2$ are independent of m. Then the concept class*

$$C_1 \wedge C_2 \wedge \cdots \wedge C_k = \{f_1 \wedge f_2 \wedge \cdots \wedge f_k \mid f_i \in C_i\}$$

is learnable in time $O(\gamma k \rho^d (t + \rho q))$ and

$$\rho = O(k^2 qd \log(kqd\gamma^{1/d} \log \gamma))$$

equivalence queries where $q = \sum_i q_i$, $t = \sum_i t_i$, $\gamma = \max_i \gamma_i$ and $d = \max_i d_i$.

In particular, when each H_i has polynomial size shatter coefficient then $C_1 \wedge C_2 \wedge \cdots \wedge C_k$ is learnable in time $O(\gamma k \rho_0^d (t + \rho_0 q))$ and

$$\rho = O(k^2 q \log(kq\gamma))$$

equivalence queries.

Proof Sketch. Consider $C = \cup_i C_i$ and $H = \cup_i H_i$. Then C is learnable in time t and q equivalence queries. Now since

$$\mathcal{S}(H_1 \cup \cdots \cup H_k, m) \le \sum_{i=1}^{k} \gamma_i m^{d_i} \le k\gamma m^d,$$

by Theorem 2 the result follows. ■

We now show the above results with WINNOW2, [9].

For any halfspace $f(x) = [a_1 x_1 + \cdots + a_n x_n \ge b]$ let $\alpha(f)$ be the minimal $\sum_{i=1}^n \mu_i/\delta^2$ such that for all $(x_1, \dots, x_n) \in \{0,1\}^n$ we have

$$\sum_{i=1}^{n} \mu_i x_i \ge 1 \qquad \text{if} \quad f(x_1, \dots, x_n) = 1$$

and

$$\sum_{i=1}^{n} \mu_i x_i \le 1 - \delta \qquad \text{if} \quad f(x_1, \dots, x_n) = 0.$$

Then we have

Theorem 3. *Let HS_α be the set of halfspaces f over $V = \{x_1, x_2, \cdots\}$ with $\alpha(f) \le \alpha$. Let C be a concept class that is learnable from H in time t and q equivalence queries. Suppose $\mathcal{S}(H, m) \le \gamma m^d$ for some d and $\gamma \ge 2$ that are independent of m. Then $\mathrm{HS}_\alpha(C)$ is learnable in time $O(\gamma \rho^d (t + \rho q))$ and*

$$\rho = O(\alpha kqd \log(\alpha kqd\gamma^{1/d} \log \gamma))$$

equivalence queries.

In particular, when H has polynomial size shatter coefficient then $\mathrm{HS}_\alpha(C)$ is learnable in time $O(\gamma \rho_0^d (t + \rho_0 q))$ and

$$\rho_0 = O(\alpha kq \log(\alpha kq\gamma))$$

equivalence queries.

As an application of Theorem 3 consider the concept class of boolean functions on strings that is a threshold of weighted substrings. For $w \in \{0,1\}^{\leq n}$ let $f_w : \{0,1\}^n \rightarrow \{0,1\}$ be the function $f_w(x) = 1$ if and only if x contains w as a substring, i.e., there is i such that $w = x_i x_{i+1} \cdots x_{i+|w|-1}$. Consider the concept class W of all f_w for all strings w over $\{0,1\}$. Then the concept class of threshold of weighted substrings is $\mathrm{HS}_\alpha(W)$.

We show

Theorem 4. *Let* HS_α *be the set of halfspaces* f *over* $V = \{x_1, x_2, \cdots\}$ *with* $\alpha(f) \leq \alpha$. *Then* $\mathrm{HS}_\alpha(W)$ *is learnable in time* $O(\gamma \rho_0^d (t + \rho_0 q))$ *and asks*

$$\rho_0 = O(\alpha k n^2 \log(\alpha k n))$$

equivalence queries.

Proof. Since m strings can have at most $n^2 m$ different substrings, we have $\mathcal{S}(W, m) \leq n^2 m$. Now it is easy to see that W is learnable from W with $q \leq n^2$ equivalence queries. Then with Theorem 3 the result follows. ■

In the full paper we give more results on learning k-CNF(C) and show how to handle errors in the answers to the equivalence queries.

Acknowledgement. We would like to thank Adam Klivans for pointing to us some of the work done in the area.

References

1. D. Angluin. Queries and Concept Learning. *Machine Learning*, 2, 319–342, 1988.
2. P. Auer, S. Kwek, W. Maass, M. K. Warmuth. Learning of Depth Two Neural Networks with Constant Fan-in at the Hidden Nodes. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(55), 2000.
3. A. Blum. Rank-r Decision Trees are a Subclass of r-Decision Lists. *Inf. Process. Lett.* 42(4), 183–185, 1992.
4. N. H. Bshouty. A Subexponential Exact Learning Algorithm for DNF Using Equivalence Queries. *Inf. Process. Lett.* 59(1), 37–39 1996.
5. N. H. Bshouty. A new Composition Theorem for Learning Algorithms. In *Proceedings of the 30th annual ACM Symposium on Theory of Computing (STOC)*, 583–589, 1998.
6. A. Ehrenfeucht, D. Haussler. Learning Decision Trees from Random Examples. *Inf. Comput.* 82(3), 231–246, 1989.
7. M. Kearns, M. Li, L. Pitt and L. Valiant. On the learnability of boolean formulae. *In Proceeding of the 19th ACM Symposium on the Theory of Computing*, 285–294, 1987.
8. A. R. Klivans, R. A. Servedio. Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. *J. Comput. Syst. Sci.* 68(2), 303–318, 2004.
9. N. Littlestone. Learning when irrelevant attributes abound. A new linear-threshold algorithm. *Machine Learning*, 2, 285–318, 1988.
10. W. Maass and M. K. Warmuth. Efficienct Learning with Virtual Threshold Gates. *Information and Computation*, 141, 66–83, 1998.

11. L. Pitt and M. K. Warmuth. Prediction-preserving reducibility. *Journal of Computer and System Science*, 41(3), 430–467, 1990.
12. N. Sauer. On the dencity of families of sets. *J. Combinatorial Theory*, Ser. A 13, 145–147, 1972.
13. J. Tarui, T. Tsukiji. Learning DNF by Approximating Inclusion-Exclusion Formulae.*IEEE Conference on Computational Complexity*, 215–220, 1999.
14. L. G. Valiant. A theory of the learnable. *Communication of the ACM*, 27(11), 1984.