Revising Threshold Functions

Robert H. Sloan [a,1] Balázs Szörényi [b,2] György Turán [c,b,1]

[a] *Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7053, USA*

[b] *Research Group on Artificial Intelligence, Hungarian Academy of Sciences and University of Szeged, Szeged, Hungary-6720*

[c] *Department of Mathematics, Statistics and Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7045, USA*

Abstract

A revision algorithm is a learning algorithm that identifies the target concept, starting from an initial concept. Such an algorithm is considered efficient if its complexity (in terms of the resource one is interested in) is polynomial in the syntactic distance between the initial and the target concept, but only polylogarithmic in the number of variables in the universe. We give an efficient revision algorithm in the model of learning with equivalence and membership queries for threshold functions, and some negative results showing, for instance, that threshold functions cannot be revised efficiently from either type of query alone. The algorithms work in a general revision model where both deletion and addition type revision operators are allowed.

1 Introduction

Computationally efficient learnability has been studied from many different angles in computational learning theory for the last two decades, for example, in both the PAC and query learning models, and by measuring complexity in terms of sample size, the number of queries, or running time. Attribute-efficient learning algorithms are required to be efficient (polynomial) in the

Email addresses: sloan@uic.edu (Robert H. Sloan), szorenyi@inf.u-szeged.hu (Balázs Szörényi), gyt@uic.edu (György Turán).

URL: www.cs.uic.edu/~sloan (Robert H. Sloan).

[1] Research supported in part by by the National Science Foundation under grants CCR-0100336 and CCF-0431059.

[2] Part of this work was done while Szörényi was visiting at University of Illinois at Chicago.

number of relevant variables, and "super-efficient" (polylogarithmic) in the total number of variables [2, 3]. It is argued that practical and biologically plausible learning algorithms need to be attribute efficient.

A related notion, *efficient revision algorithms*, originated in machine learning [9, 13, 16, 21], and has received some attention in computational learning theory as well. A revision algorithm is applied in a situation where learning does not start from scratch, but there is an initial concept available, which is a reasonable approximation of the target concept. The standard example is an initial version of an expert system provided by a domain expert. The efficiency criterion in this case is to be efficient (polynomial) in the *distance* from the initial concept to the target (whatever distance means; we will return to this issue shortly), and to be "super-efficient" (polylogarithmic) in the total size of the initial formula. Again, it is argued that this is a realistic requirement, as many complex concepts can only be hoped to be learned efficiently if a reasonably good initial approximation is available. The notion of distance usually considered is a syntactic one: the number of edit operations that need to be applied to the initial representation in order to get a representation of the target. The particular edit operations considered depend on the concept class. Intuitively, attribute-efficient learning is a special case of efficient revision, when the initial concept has an empty representation. In machine learning, the study of revision algorithms is referred to as theory revision; detailed references to the literature are given in Wrobel's overviews of theory revision [26, 27] and also in our recent papers [5, 6].

It is a general observation both in practice and in theory that edit operations that delete something from the initial representation are easier to handle than those that add something to it. In practice, of course, revisions that add are just as important as those that delete; there is no reason to believe that small errors in, say, an expert system, would be one sided.

The theoretical study of revision algorithms was initiated by Mooney [12] in the PAC framework. We have studied revision algorithms in the model of learning with equivalence and membership queries [5, 6] and in the mistake-bound model [19]. Formal definitions for query model learning, that is, learning from equivalence and membership queries [1], are given in Section 2. For purposes of this introduction, a membership query asks whether an instance is positive (i.e., belongs to the target concept) or negative (does not belong to the target concept), and an equivalence query asks whether a proposed concept is correct, and a counterexample is returned if the proposed concept is not correct.

It seems to be an interesting general question whether attribute-efficiently learnable classes can also be revised efficiently. Our previous work answers this question negatively. Monotone DNF *can* be learned by an attribute-efficient

learning algorithm [2], but our previous work on revising DNF [6] shows, among other things, that monotone DNF cannot be revised efficiently. We did obtain efficient revision algorithms for monotone DNF with a bounded number of terms when both deletion and addition type revisions are allowed. However, we also showed that efficient revision of general (or even monotone) DNF is not possible, even with only deletion type revisions.

Two classes of Boolean functions that seem to be important in practice for AI applications are Horn sentences, the foundation of rule-based systems, and threshold functions, the foundation of neural nets. Previously we have given revision algorithms for Horn sentences if only deletion type revisions are permitted [5], and for certain restricted classes of Horn sentences with both types of revisions [4]. In this article, we consider threshold functions, and we give an efficient revision algorithm for threshold functions with both types of revisions.

We have also given revision algorithms (with both types of revisions) for two other classes that have attribute-efficient learning algorithms: for parity functions in [6] and for projective DNF in [19]. Projective DNF is a class of DNF introduced by Valiant [23], as a special case of his projective learning model, and as part of a framework to formulate expressive and biologically plausible learning models. In biological terms revision may be relevant for learning when some information is hard-wired from birth; see, e.g., Pinker [14] for recent arguments in favor of hereditary information in the brain.

Valiant showed that projective DNF are attribute-efficiently learnable in the mistake-bound model, and we extended his result by showing that they are efficiently revisable. Our algorithm was based on showing that a natural extension of the Winnow algorithm is in fact an efficient revision algorithm for disjunctions even in the presence of attribute errors.

Valiant's related models [24, 25] also involve threshold functions, and as threshold functions are also known to be attribute-efficiently learnable, this raises the question whether threshold functions can be revised efficiently. Threshold functions (also called Boolean threshold functions or zero-one threshold functions in the literature) form a much studied concept class in computational learning theory. Winnow is an attribute-efficient mistake-bounded learning algorithm [10]. Hegedűs [7] gave $\Theta(n)$ upper and lower bounds (n is the total number of variables) for the number of queries needed to learn threshold functions in the query model; the algorithm uses only membership queries.

Attribute-efficient proper query learning algorithms are given in Uehara et al. [22] and Hegedűs and Indyk [8]. Further related results are given in [15, 17, 20].

In this paper we present results for the revision of threshold functions, in the general revision model allowing both deletions and additions (more precise definitions are given in Section 2). We use the model of learning with

membership and equivalence queries.

Our main result is a revision algorithm for threshold functions using $O(\mathrm{dist}(\varphi, \psi) \cdot \log n)$ queries (Theorem 5), where dist is the revision distance, defined formally in Section 2. In this algorithm the pattern mentioned above is reversed, and it turns out to be easier to handle additions than deletions. It is also shown that both query types are necessary for efficient revision, and that the query complexity of the algorithm is essentially optimal up to order of magnitude. Another interesting point is that the natural extension of Winnow mentioned above does not work in this more general context.

Organization of paper Preliminaries are given in Section 2, and the main revision algorithm for threshold functions, including analysis, in Section 3. Some lower bounds on the problem are given, and one open problem is presented in Section 4. Finally an example run of the revision algorithm is given in the Appendix.

2 Preliminaries

We use standard notions from propositional logic such as variable, literal, term (or conjunction), clause (or disjunction), etc. The set of variables for n-variable formulas and functions is $X_n = \{x_1, \ldots, x_n\}$. (In this paper, n will always be the total number of variables.) *Instances* or *vectors* are elements $\mathbf{x} \in \{0,1\}^n$. When convenient we treat $\mathbf{x}$ as a subset of $[n] = \{1, \ldots, n\}$ or X_n, corresponding to the components, resp. the variables, which are set to true in $\mathbf{x}$. Given a set $Y \subseteq [n]$, we write $\chi_Y = (\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$ for the characteristic vector of Y. We write $\mathbf{x} = (x_1, \ldots, x_n) \leq \mathbf{y} = (y_1, \ldots, y_n)$ if $x_i \leq y_i$ for every $i = 1, \ldots, n$.

An n-variable *threshold function* TH^t_U is specified by a set $U \subseteq [n]$ and a *threshold* $0 \leq t \leq n$, such that for a vector $\mathbf{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ it holds that $\mathrm{TH}^t_U(\mathbf{x}) = 1$ if at least t of the variables with subscripts in U are set to 1 in $\mathbf{x}$; otherwise $\mathrm{TH}^t_U(\mathbf{x}) = 0$. In other words, $\mathrm{TH}^t_U(\mathbf{x}) = 1$ iff $\sum_{i=1}^n \alpha_i x_i \geq t$, where $\chi_U = (\alpha_1, \ldots, \alpha_n)$. We say that S is a *positive* (resp., *negative*) set if χ_S is a positive (resp., negative) example of the target threshold function. (As the number of variables is clear from the context, we do not mention it in the notation.) Note that for every non-constant threshold function its set of relevant variables and its threshold are well defined, thus every non-constant function has a unique representation. The variables with indices in U (resp., outside of U) are the *relevant* (resp., *irrelevant*) variables of TH^t_U. As noted in the introduction, functions of this type are also called Boolean threshold functions and 0-1-threshold functions, in order to distinguish them

from the more general kind of threshold functions where the coefficients α_i can be arbitrary real numbers. We simply call them threshold functions, as we only consider this restricted class.

A set S is *maximal negative* (or critical) for threshold function TH_U^t if $|S \cap U| = t-1$; and *minimal positive* for TH_U^t if $|S \cap U| = t$.

We use the standard model of membership and equivalence queries (with counterexamples), denoted by MQ and EQ [1]. The unknown Boolean function to be learned is called the *target concept* (in this paper, always a threshold function that is a revision of a given threshold function). In an equivalence query, the learning algorithm proposes a *hypothesis*, a concept h, and the answer depends on whether $h \equiv c$, where c is the target concept. If so, the answer is "correct", and the learning algorithm has succeeded in its goal of exact identification of the target concept. Otherwise, the answer is a *counterexample*, any instance $\mathbf{x}$ such that $c(\mathbf{x}) \neq h(\mathbf{x})$. In this paper we consider *proper* equivalence queries, meaning that the queried hypotheses must also be taken from the given concept class—in our case the class of Boolean threshold functions. When the learning algorithm makes a membership query on instance $\mathbf{x}$, denoted $\mathrm{MQ}(\mathbf{x})$, it gets back the value $c(\mathbf{x})$, where c is the target concept.

Given the above, we can state the following proposition which we use implicitly throughout:

Proposition 1 *If S is maximal negative for $\psi = \mathrm{TH}_U^t$, then for every $Z \subseteq X_n \setminus S$ it holds that Z contains at least one variable in U (i.e., relevant variable of ψ) iff $\mathrm{MQ}(\chi_{S \cup Z}) = 1$.*

2.1 Revision

The *revision distance* between a representation φ of an initial Boolean function (or concept) and a concept c is defined to be the minimum number of applications of a specified set of syntactic revision operators to φ needed to obtain a representation of c (from a specified set of representations; in this paper threshold functions are always represented by specifying the set of relevant variables and the threshold). The revision operators may depend on the concept class one is interested in. Usually, a revision operator can either be *deletion-type* or *addition-type*.

In the case of threshold functions, deletions mean deleting a relevant variable and additions mean adding a new relevant variable. In the *general model* for this class we also allow the *modification of the threshold*. We consider the modification of the threshold by any amount to be a single operation (as opposed to changing it by one); as we are going to prove upper bounds, this only

makes the results stronger. Thus, for example, the revision distance between $\varphi = \mathrm{TH}^1_{\{x_1,x_2,x_4\}}$ and $\mathrm{TH}^3_{\{x_1,x_2,x_3,x_5\}}$ is 4 in the general model.

We use $\mathrm{dist}(\varphi, \psi)$ to denote the revision distance from φ to ψ whenever the revision operators are clear from context.

A *revision algorithm* for a (representation of a) function φ has access to membership and equivalence oracles for an unknown target concept and must return some representation of the target concept. Our goal is to find revision algorithms whose query complexity is polynomial in $d = \mathrm{dist}(\varphi, \psi)$, but at most *polylogarithmic* in n, the number of variables in the universe.

We state only query bounds in this paper; all our revision algorithms run in polynomial time, given access to the membership and equivalence query oracles.

3 Revising threshold functions: Algorithm

We present a threshold revision algorithm ReviseThreshold. The overall revision algorithm is given as Algorithm 1, using the procedures described in Algorithms 3 and 4. Throughout this section, let the initial function be $\varphi = TH^t_U$ and the target function be $\psi = TH^\theta_R$. Algorithm ReviseThreshold has three main stages. First we identify all the variables that are irrelevant in φ but relevant in ψ (Algorithm FindAdditions). Then we identify all the variables that are relevant in φ but irrelevant in ψ (Algorithm FindDeletions). Finally, we determine the target threshold. (In our pseudocode this third step is built into Algorithm FindDeletions as the last iteration, after the set of relevant variables of the target function is identified.)

A sample run of the algorithm is given in Appendix A. It is broken up in two subsections, reflecting the above described partitioning of the original task.

Algorithm 1 The procedure ReviseThreshold(φ), where $\varphi = \mathrm{TH}^t_U$.

1: Use 2 MQ's to determine if target is constant 0 or 1; if so **return**
2: $V :=$ FindAdditions(U)
3: $\psi :=$ FindDeletions($U \cup V$)
4: **return** ψ

Before getting into further details, we need to point out an additional subroutine. Our revision algorithm frequently uses a kind of binary search, often used in learning algorithms involving membership queries, presented as Algorithm 2. The starting points of the binary search are two instances, a negative instance **neg** and a positive instance **pos** such that **neg** $\leq$ **pos**. The algorithm returns two items: the first is a set of variables that when added to **neg** make

a positive instance; the second is a variable that is maximal negative in the sense that the first component plus **neg** becomes a negative instance if that variable is turned off.

Algorithm 2 BINARYSEARCH(**neg**, **pos**).

Require: MQ(**neg**) = 0 and MQ(**pos**) = 1 and **neg** $\leq$ **pos**

1: $\mathbf{neg}_0 :=$ **neg**
2: **while neg** and **pos** differ in more than 1 position **do**
3: Partition **pos** $\setminus$ **neg** into approximately equal-size sets d_1 and d_2.
4: Put **mid** := **neg** with positions in d_1 switched to 1
5: **if** MQ(**mid**) = 0 **then**
6: **neg** := **mid**
7: **else**
8: **pos** := **mid**
9: **end if**
10: **end while**
11: $v :=$ the one variable on which **pos** and **neg** differ
12: **return** $((\mathbf{pos} \setminus \mathbf{neg}_0), v)$

3.1 Correctness and analysis

First we analyze algorithm FINDADDITIONS (Algorithm 3), which is responsible for finding all missing relevant variables.

Lemma 2 *Let R be the relevant variables of the nonconstant target function. If Algorithm* FINDADDITIONS *is called with input $U \subseteq X_n$, then it returns $R \setminus U$, using $O(|R \setminus U| \log n)$ queries.*

PROOF. The algorithm stores the uncertain but potentially relevant variables in the set *Potentials* (thus *Potentials* is initially set to $X_n \setminus U$). The procedure first determines a set $Base \subseteq U$ such that *Base* is negative, and $Base \cup Potentials$ is positive (unless *Potentials* contains no relevant variables—in which case there are no new relevant variables used by ψ, so we quit in Line 8).

Then the search for new relevant variables starts. We repeatedly use BINARYSEARCH($Base, Base \cup Potentials$) to find one relevant variable, and then remove this variable from *Potentials*. After removing a certain number of relevant variables from *Potentials*, the instance $Base \cup Potentials$ must become minimal positive. After reaching this point, we do not only remove any newly found relevant variables from *Potentials*, but we also add them to the set *Base*. From this point on, it holds that $|(Base \cup Potentials) \cap R| = \theta$. Thus

Algorithm 3 The procedure FINDADDITIONS(U)

Require: the target function is not constant

1: $Potentials := X_n \setminus U$
2: **if** $\mathrm{MQ}(\chi_U) = 0$ **then**
3: $\quad Base := U$
4: **else**
5: $\quad (Base, x) :=$ BINARYSEARCH$(\emptyset, U)$
6: $\quad Base := Base \setminus \{x\}$
7: $\quad$ **if** $\mathrm{MQ}(\chi_{Base \cup Potentials}) = 0$ **then**
8: $\quad\quad$ **return** $\emptyset$
9: $\quad$ **end if**
10: **end if**
11: $NewRelevants := \emptyset$
12: **repeat**
13: $\quad (Y, y) :=$ BINARYSEARCH$(Base, Base \cup Potentials)$
14: $\quad NewRelevants := NewRelevants \cup \{y\}$
15: $\quad Potentials := Potentials \setminus \{y\}$
16: $\quad$ **if** $\mathrm{MQ}(\chi_{Base \cup Potentials}) = 0$ **then**
17: $\quad\quad Base := Base \cup \{y\}$
18: $\quad$ **end if**
19: **until** $\mathrm{MQ}(\chi_{Base}) = 1$
20: **return** $NewRelevants$

the indicator that the last relevant variable has been removed from *Potentials* is that *Base* becomes positive ($\mathrm{MQ}(\chi_{Base}) = 1$).

As BINARYSEARCH always uses at most $\lceil \log_2 n \rceil$ membership queries per call, and one addition requires one call to BINARYSEARCH and at most two other membership queries are made initially, the stated query complexity follows. □

Now we turn to the discussion of procedure FINDDELETIONS (Algorithm 4), which finds all the irrelevant variables that appear in the initial hypotheses. The procedure uses a function called MAKEEVEN, presented as Algorithm 5. MAKEEVEN makes at most two queries; its main task is to move variables around to ensure needed conditions, mostly parity, on certain sets. A more detailed prose description of its behavior is given in the proof of Lemma 3.

Lemma 3 *If the target function $\psi = TH^{\theta}_{R}$ is not constant and if $R \subseteq H \subseteq X_n$, then if Algorithm* FINDDELETIONS *is called with input H, it returns ψ, using $O(|H \setminus R| \log n)$ queries.*

Algorithm 4 The procedure FINDDELETIONS(H)

Require: $R \subseteq H$ (R = relevant variables in target)

1: **if** $(\mathbf{x}_P := \mathrm{EQ}(\mathrm{TH}_H^{|H|})) = YES$ **then**
2: **return** $\mathrm{TH}_H^{|H|}$
3: **end if**
4: **if** $(\mathbf{x}_N := \mathrm{EQ}(\mathrm{TH}_H^{1})) = YES$ **then**
5: **return** TH_H^{1}
6: **end if**
7: $P := \mathbf{x}_P \cap H$; $N := \mathbf{x}_N \cap H$
8: $\ell := 1; u := |H|$
9: **while** $u > \ell + 1$ **do**
10: $m := \lceil (u + \ell)/2 \rceil$
11: **if** $(\mathbf{x} := \mathrm{EQ}(\mathrm{TH}_H^{m})) = YES$ **then**
12: **return** TH_H^{m}
13: **end if**
14: $\mathbf{x} := \mathbf{x} \cap H$ {Variables not in H are irrelevant}
15: **if** $\mathbf{x}$ is a positive counterexample **then**
16: $P := \mathbf{x}$ and $u := m$
17: **else**
18: $N := \mathbf{x}$ and $\ell := m$
19: **end if**
20: **end while**
21: $(P, p) :=$ BINARYSEARCH$(\emptyset, P)$
22: $Base := P \cap N$, $N' := N \setminus Base$, $P' := P \setminus Base$
23: **while** $|P'| > 1$ **do**
24: *changedH* :=MAKEEVEN$(Base, N', P', p, H)$ {Uses at most 2 MQs}
25: **if** *changedH* **then**
26: **goto** Line 1
27: **end if**
28: Let N_0, N_1 (resp. P_0, P_1) be an equal-sized partition of N' (resp. P')
29: Ask $\mathrm{MQ}(\chi_{Base \cup N_j \cup P_k})$ for $j, k = 0, 1$
30: Let j and k be indices s.t. $\mathrm{MQ}(\chi_{Base \cup N_j \cup P_k}) = 0$ {such j and k exist}
31: $Base := Base \cup P_k$, $P' := P_{1-k}$, $N' := N_j$
32: **end while**
33: $H := H \setminus N'$
34: **goto** Line 1

PROOF. First consider the case where no variables need to be deleted from H. If the threshold is either 1 or $|H|$, this will be found by one of the two initial equivalence queries to those two threshold functions. If the threshold is some value in between, then it will be found by a binary search over threshold values carried out by the first while loop. Then the correct threshold function is returned at Line 12.

Algorithm 5 Function MAKEEVEN($Base, N', P', p, H$)

1: $Test := (Base \cup P') \setminus \{p\}$
   {For any $i \in N'$, $\mathrm{MQ}(\chi_{Test \cup \{i\}}) = 1$ iff i is relevant}
2: **if** $|P'|$ is odd **then**
3: Choose $x_a \in P'$ arbitrarily and move x_a from P' to $Base$
4: Choose $x_i \in N'$ arbitrarily and move x_i from N' to $Base$
5: **if** $\mathrm{MQ}(\chi_{Test \cup \{i\}}) \neq 1$ **then** {x_i irrelevant}
6: $H := H \setminus \{x_i\}$
7: **return** *true* {H was modified}
8: **end if**
9: **end if**
10: **if** $|N'|$ is odd **then**
11: Choose $x_i \in N'$ arbitrarily and move x_i from N' to $Base$
12: **if** $\mathrm{MQ}(\chi_{Test \cup \{i\}}) \neq 1$ **then** {x_i irrelevant}
13: $H := H \setminus \{x_i\}$
14: **return** *true* {H was modified}
15: **end if**
16: **end if**
17: **return** *false* {H was not modified.}

Otherwise, there are some variables that need to be deleted. In this case, our short-term goal is to find two sets of variables N and P such that

$$|N| \geq |P|, \text{ and } N \text{ is negative and } P \text{ is positive for } \mathrm{TH}^{\theta}_{R} \ . \tag{1}$$

The two initial equivalence queries must have assigned P to be a positive counterexample to TH^1_H and N to be a negative counterexample to $\mathrm{TH}^{|H|}_H$. In the binary search over threshold values in the **while** loop at Lines 9–20, N is always assigned negative counterexamples from equivalence queries and P is always assigned positive counterexamples from equivalence queries.

Now we need to argue that at the end of that binary search (i.e., after Line 20), we will have $|N| \geq |P|$. Consider the last time that N is updated. (This could be either when $\ell = 1$ before the **while** loop or inside the **while** loop.) At that update, we set N to be the variables from the negative counterexample that are not known to be irrelevant. That is, we set N to be $\mathbf{x} \cap H$, where $\mathbf{x}$ was the counterexample from the equivalence query to TH^m_H (or to TH^1_H if this was before the **while** loop). Since we used a *negative* counterexample it must be that $\mathrm{TH}^m_H(\chi_N) = 1$. Thus we know that $|N| \geq m$. In the control of the binary search over threshold values, the lower bound ℓ now becomes m, and ℓ is not updated again. Thus this value of ℓ is the value of ℓ after the loop has ended, and $|N| \geq \ell$ from now on.

Similar conditions hold for P and u, the upper bound in the control of the binary search. After the last update to P, it must be that $|P| < m$ (since P is

a positive counterexample), u is updated to be this m, and u is not updated again. Thus $|P| < u$.

When the **while** loop terminates, $u \leq \ell + 1$. Since $|P| < u \leq \ell + 1$, we have $|P| \leq \ell$. Since $|N| \geq \ell$, we now have Equation (1).

Now we want to use N and P to construct three sets with what we call the *"key property:"*

Key property: *A triple of sets of variables $(Base, N', P')$ satisfies the key property for (target) threshold function TH_θ^R if the sets are pairwise disjoint, and it holds that*

- *$Base \cup N'$ is negative,*
- *$|(Base \cup P') \cap R| = \theta$ (i.e., $Base \cup P'$) is a minimal positive set), and*
- *$|N'| \geq |P'|$.*

Given N and P satisfying Equation (1), in Line 21 we make make P the set returned by BinarySearch$(\emptyset, P)$, which makes P a minimal positive set. We then set $Base = N \cap P$, and $P' = P \setminus Base$ and $N' = N \setminus Base$. The key property must hold for this triple: $N = Base \cup N'$ is negative; $P' = Base' \cup P$ is a minimal positive set, and it must be that $|N'| \geq |P'|$.

The following claim gives two important features of the key property.

Claim 4 a) *If $(Base, N', P')$ satisfies the key property, then N' contains an irrelevant variable and P' contains a relevant variable.*

b) *If $(Base, N', P')$ satisfies the key property and $|P'| = 1$, then every element of N' is irrelevant.*

Our overall goal now is to find at least one of the irrelevant variables in N' and delete it. From now on we maintain the key property among the three sets, but in a way that in each iteration the size of N' and P' gets halved. For this we split up N' (respectively P') into two equal-sized disjoint subsets N_1 and N_2 (resp. P_1 and P_2). When both $|N'|$ and $|P'|$ are even then we can do this without any problem; otherwise we have to make some adjustments to N' and/or to P', that will be taken care of by procedure MakeEven, which we will describe presently.

Assume for now that both $|N'|$ and $|P'|$ are even. Let $\theta' = \theta - |R \cap Base|$. We have $|R \cap (N_1 \cup N_2)| < \theta'$ and $|R \cap (P_1 \cup P_2)| = \theta'$. Thus for some $j, k \in \{0, 1\}$ we have $|R \cap (N_j \cup P_k)| < \theta'$ (equivalently $\mathrm{MQ}(\chi_{Base \cup N_j \cup P_k}) = 0$). Note that the sets $Base := Base \cup P_k$, $N' := N_j$ and $P' := P_{1-k}$ still have the key property, but the size of N' and P' is reduced by half. Thus after at most $\log n$ steps P' is reduced to a set consisting of a single (relevant) variable. Thus N' is a

nonempty set of irrelevant variables (part *b)* of Claim 4).

Finally, the function MakeEven$(Base, N', P')$ works as follows. Its job is to move variables among sets so as to preserve the key property for $Base$, N', and P', while making both N' and P' have even size. Sometimes instead, however, it will remove an irrelevant variable from H—in this case it returns *true* and its caller restarts with the smaller H.

First *MakeEven* checks whether $|P'|$ is odd, and if so, it moves an arbitrary element x_a of P' to $Base$. Note that if x_a was relevant, this action might turn $Base \cup N'$ into a positive set; thus the key property might be violated; so an arbitrary element x_i will also be removed from N'. If x_i is irrelevant (which can be tested using set $Test$ defined at Line 1), MakeEven removes it from H and immediately returns true, so the overall search can be restarted.

Otherwise (i.e, if x_i is relevant) the key property holds for the new triple $(Base, N', P')$, and $|P'|$ is even. Then MakeEven checks if $|N'|$ is odd, and if so, an arbitrary x_i gets removed from N'; again we have the same check whether x_i is irrelevant.

If *MakeEven* returns false (no irrelevant x_i was removed from H), then the resulting triple will also have the key property.

Now we give the complexity analysis.

For each deletion found, we can require first $2 + \lceil \log_2 n \rceil$ equivalence queries to get the sets N and P, and then one call to BinarySearch to make P a minimal positive set. We next iterate, shrinking both $|P'|$ and $|N'|$ by half in each iteration, at most $\lceil \log_2 n \rceil$ times. In each such iteration we make at most 7 membership queries. Thus (as BinarySearch always uses at most $\lceil \log_2 n \rceil$ membership queries per call) the deletions require at most $O(|H \setminus R| \log n)$ queries. □

Now we can state the main result of the section.

Theorem 5 ReviseThreshold *is a threshold function revision algorithm of query complexity* $O(\mathrm{dist}(\varphi, \psi) \log n)$*, where* φ *is the initial function and* ψ *is the target function.*

PROOF. First, two membership queries are used to determine if the target is either of the two constant Boolean functions. For nonconstant functions, the complexity and the correctness follow from Lemmas 2 and 3. □

4 Lower bounds and an open problem

In this section, we show that both types of queries are needed for the efficient revision of threshold functions, and that the query complexity of our algorithm is essentially optimal up to order of magnitude. The first result shows that efficient revision is not possible with membership queries, even if we allow a restricted type of equivalence queries as well, and the second result shows that efficient revision is not possible with equivalence queries alone.

Theorem 6 *Efficient revision of threshold functions is not possible if both membership and equivalence queries can be used, but the equivalence queries must always use the threshold value of the initial function (which is guaranteed to be the threshold of the target as well).*

PROOF. Let the initial function be $\mathrm{TH}^{n-1}_{\{x_1,\ldots,x_n\}}$.

Let $\psi_i = \mathrm{TH}^{n-1}_{\{x_1,\ldots,x_n\}\setminus\{x_i\}}$ for $1 \le i \le n$. Initially the adversary places every ψ_i in a set Ψ of possible target concepts.

The adversary answers the learner's membership query $\mathrm{MQ}(\chi_U)$ for some $U \subseteq \{x_1, \ldots, x_n\}$ as follows:

- "no", if $|U| < n-1$.
- "yes", if $|U| = n$ or if $\Psi = \{\mathrm{TH}^{n-1}_U\}$
- "no", if neither of the above applies. Also, set $\Psi = \Psi \setminus \{\psi_i\}$ for i with $\{x_i\} = \{x_1, \ldots, x_n\} \setminus U$.

The adversary answers the learner's equivalence query $\mathrm{EQ}(\mathrm{TH}^{n-1}_U)$ for some $U \subseteq \{x_1, \ldots, x_n\}$ as follows:

- If $|U| < n-1$ (i.e., the hypothesis is the everywhere-false function) then return vector 1^n as a positive counterexample.
- If $|U| = n$, then for some $\psi_i \notin \Psi$ present the vector that is 0 in position i and 1 elsewhere as a positive counterexample.

  If no ψ_i was previously removed from Ψ, then present 01^{n-1} as a positive counterexample, and remove ψ_1 from Ψ.
- If $U = \{x_1, \ldots, x_n\} \setminus \{x_i\}$ and $|\Psi \setminus \psi_i| \ge 1$, then present χ_U as a negative counterexample, and remove ψ_i from the set Ψ.
- If none of the above applies (thus $\Psi = \{\mathrm{TH}^{n-1}_U\}$), return "yes".

It follows by a standard case analysis that $|\Psi|$ is decreased by at most one after each query. Thus the learner must make at least n queries, although the revision distance is 1. □

Theorem 7 *Efficient revision of threshold functions is not possible using only equivalence queries.*

PROOF. Set $n = 2k$. We give an adversary argument with initial function $\mathrm{TH}^k_{\{x_1,\ldots,x_n\}}$, where the universe of variables is $\{x_1, \ldots, x_n\}$. For $k+1 \leq i \leq n$, let $\psi_i = \mathrm{TH}^k_{\{x_1,\ldots,x_n\}\setminus\{x_i\}}$. The target will be one of the ψ_i, and the adversary initializes a set Ψ to be all the ψ_i.

In response to the learner's query $\mathrm{EQ}(\mathrm{TH}^\ell_U)$, answer "no" (except where specified otherwise below), and

- if $\ell < k$:
  - If $|U| \geq \ell$ present $\chi_{U'}$ as a negative counterexample, where U' is any subset of U with cardinality ℓ.
  - Otherwise present 1^k0^k as a positive counterexample.
- If $\ell > k$, present 1^k0^k as a positive counterexample.
- If $\ell = k$:
  - In case $U \supseteq \{x_{k+1}, \ldots, x_n\}$, present 0^k1^k as a negative counterexample.
  - Otherwise, if $\{x_1, \ldots, x_k\} \not\subseteq U$ present 1^k0^k as a positive counterexample.
  - Now it must be that U contains all of $\{x_1, \ldots, x_k\}$, and is missing at least one of $\{x_{k+1}, \ldots, x_n\}$.

    The counterexample returned with a "no" answer should be $\chi_{\{2,\ldots,k\}\cup\{i\}}$ for some "missing" x_i.

    In particular, if there is a missing x_i such that $\psi_i \notin \Psi$, use that value of i. If not, and if $|\Psi| > 1$, then use any i, and also remove ψ_i from Ψ.

    Otherwise, return "yes."

So the revision algorithm must make at least n queries when the revision distance is only 1. □

Now we show that the query bound of algorithm ReviseThreshold cannot be improved for small values of d (i.e., constant d), and cannot be much improved in general. We gave a revision algorithm with query complexity $O(d \log n)$; we give here the close lower bound of $\Omega(d \log(n/d))$. (We think that the first one is closer to the real answer)

Proposition 8 *There is a threshold function φ such that the number of membership and equivalence queries needed to find a distance d revision of φ is $\Omega(d \log \frac{n}{d})$.*

PROOF. The VC-dimension is a lower bound on the number of queries needed to learn a function in the query model (from both types of queries) [11].

Let φ be $\mathrm{TH}^1_\emptyset$. The result follows from the fact that the VC-dimension of disjunctions consisting of at most d variables is $\Omega(d \log \frac{n}{d})$ [10]. $\square$

The following result answers the question that arises naturally whenever one is learning threshold functions: why not use Winnow? After all it is one of the most successful tools for learning threshold functions. Furthermore, previously it has been successfully used for revision (see, e.g., [18, 19]). The answer is simple and somewhat surprising: under our settings, using Winnow as defined in [10] would result in an *inefficient* revision algorithm.

Proposition 9 *Winnow is not an efficient revision algorithm for threshold functions. More precisely, for any weight vector representing the initial threshold function* $\mathrm{TH}^1_{x_1,\ldots,x_n}$*, Winnow can make* n *mistakes when the target function is* $\mathrm{TH}^2_{x_1,\ldots,x_n}$*.*

PROOF. The statement follows easily, noting that the weight of each relevant variable is at least as big as the threshold used by Winnow, thus giving Winnow the negative examples $\mathbf{e}_1 = \chi_{\{x_1\}}, \ldots, \mathbf{e}_n = \chi_{\{x_n\}}$ one after another, it will evaluate to 1 for each of them. $\square$

It would be interesting to consider disjunctions of a bounded number of threshold functions in the revision model. This class is a generalization of monotone DNF with a bounded number of terms, which can be revised efficiently [6]. It is also related to the robust logic framework of Valiant [23] mentioned in the introduction.

Acknowledgements

We thank the referees of this journal article for doing an excellent, very thorough job of reviewing. They significantly improved the quality of the presentation here.

References

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

[2] A. Blum, L. Hellerstein, and N. Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. *J. of Comput. Syst. Sci.*, 50(1):32–40, 1995. Earlier version in 4th COLT, 1991.

[3] N. Bshouty and L. Hellerstein. Attribute-efficient learning in query and mistake-bound models. *J. of Comput. Syst. Sci.*, 56(3):310–319, 1998.

[4] J. Goldsmith, R. H. Sloan, B. Szörényi, and G. Turán. New revision algorithms. In *Algorithmic Learning Theory, 15th International Conference, ALT 2004, Padova, Italy, October 2004, Proceedings*, volume 3244 of *Lecture Notes in Artificial Intelligence*, pages 395–409. Springer, 2004.

[5] J. Goldsmith, R. H. Sloan, B. Szörényi, and G. Turán. Theory revision with queries: Horn, read-once, and parity formulas. *Artificial Intelligence*, 156:139–176, 2004.

[6] J. Goldsmith, R. H. Sloan, and G. Turán. Theory revision with queries: DNF formulas. *Machine Learning*, 47(2/3):257–295, 2002.

[7] T. Hegedűs. On training simple neural networks and small-weight neurons. In *Computational Learning Theory: EuroColt '93*, volume New Series Number 53 of *The Institute of Mathematics and its Applications Conference Series*, pages 69–82, Oxford, 1994. Oxford University Press.

[8] T. Hegedűs and P. Indyk. On learning disjunctions of zero-one threshold functions with queries. In *Algorithmic Learning Theory, 8th International Workshop, ALT '97, Sendai, Japan, October 1997, Proceedings*, volume 1316 of *Lecture Notes in Artificial Intelligence*, pages 446–460. Springer, 1997.

[9] M. Koppel, R. Feldman, and A. M. Segre. Bias-driven revision of logical domain theories. *Journal of Artificial Intelligence Research*, 1:159–208, 1994.

[10] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988.

[11] W. Maass and G. Turán. Lower bound methods and separation results for on-line learning models. *Machine Learning*, 9:107–145, 1992.

[12] R. J. Mooney. A preliminary PAC analysis of theory revision. In T. Petsche, editor, *Computational Learning Theory and Natural Learning Systems*, volume III: Selecting Good Models, chapter 3, pages 43–53. MIT Press, 1995.

[13] D. Ourston and R. J. Mooney. Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, 66:273–309, 1994.

[14] S. Pinker. *The Blank Slate: The Modern Denial of Human Nature*. Viking Press, 2002.

[15] L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *J. ACM*, 35(4):965–984, 1988.

[16] B. L. Richards and R. J. Mooney. Automated refinement of first-order Horn-clause domain theories. *Machine Learning*, 19:95–131, 1995.

[17] M. Schmitt. On methods to keep learning away from intractability. In *Proc. International Conference on Artifical Neural Networks (ICANN) '95*, volume 1, pages 211–216, 1995.

[18] R. H. Sloan and B. Szörényi. Revising projective DNF in the presence of noise. In *Proc. Kalmár Workshop on Logic and Computer Science*, pages 143–152, Szeged, Hungary, Oct. 2003. Dept. of Informatics, University of Szeged. Journal-length version submitted and under review. Both versions available on-line from URL `http://www.cs.uic.edu/~sloan/papers.html`.

[19] R. H. Sloan, B. Szörényi, and G. Turán. Projective DNF formulae and their revision. In *Learning Theory and Kernel Machines, 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003, Proceedings*, volume 2777 of *Lecture Notes in Artificial Intelligence*, pages 625–639. Springer, 2003.

[20] R. H. Sloan and G. Turán. Learning from incomplete boundary queries using split graphs and hypergraphs. In *Computational Learning Theory, Third European Conference, EuroCOLT '97, Jerusalem, Israel, March 1997, Proceedings*, number 1208 in Lecture Notes in Artificial Intelligence, pages 38–50. Springer, 1997.

[21] G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, 1993.

[22] R. Uehara, K. Tsuchida, and I. Wegener. Identification of partial disjunction, parity, and threshold functions. *Theoretical Computer Science*, 230:131–147, 1999.

[23] L. G. Valiant. Projection learning. *Machine Learning*, 37(2):115–130, 1999.

[24] L. G. Valiant. A neuroidal architecture for cognitive computation. *J. ACM*, 47(5):854–882, 2000.

[25] L. G. Valiant. Robust logics. *Artificial Intelligence*, 117:231–253, 2000.

[26] S. Wrobel. *Concept Formation and Knowledge Revision*. Kluwer, 1994.

[27] S. Wrobel. First order theory refinement. In L. De Raedt, editor, *Advances in ILP*, pages 14–33. IOS Press, Amsterdam, 1995.

A Appendix: A simple example run

To demonstrate the algorithm, we provide an example run. Let the universe be $\{x_i : i = 1, \ldots, 8\}$, and the initial function φ and unknown target function ψ be

$$\varphi = \mathrm{TH}^1_{\{x_1,x_2,x_4\}}$$
$$\psi = \mathrm{TH}^4_{\{x_1,x_2,x_3,x_5,x_6\}} \ .$$

First, in subsection A.1 we determine all the relevant variables that were left out from $\{x_1, x_2, x_4\}$, then in subsection A.2 we further revise our hypotheses from subsection A.1 by removing those irrelevant variables that appeared in $\{x_1, x_2, x_4\}$.

A.1 Adding the previously unknown relevant variables

Two MQ's to 00000000 and 11111111 determine that the target function is nonconstant.

We next determine the necessary additions, that is, the relevant variables from $\{x_3, x_5, x_6, x_7, x_8\}$, using Procedure FindAdditions. As $\chi_{\{x_1,x_2,x_4\}}$ is negative, $Potentials = \{x_3, x_5, x_6, x_7, x_8\}$ must contain some unknown relevant variables.

In Lines 12–19 of Procedure FindAdditions, we repeatedly use BinarySearch from $Base = \{x_1, x_2, x_4\}$ to $Base \cup Potentials$ to find one. Inside BinarySearch ask MQ(11111100), the answer is 1. Ask MQ(11111000), the answer is 1. Ask MQ(11110000), the answer is 0. The last negative and positive examples differ by the single variable x_5—thus x_5 is relevant, and is returned to FindAdditions, and FindAdditions adds x_5 to *NewRelevants*.

Now exclude the newly found relevant variable x_5 from consideration. As $\chi_{Base\cup\{x_3,x_6,x_7,x_8\}}$ is still positive, we make another similar call to BinarySearch. Ask MQ(11110100), the answer is 1. Ask MQ(11110000), the answer is 0. The last positive and negative vectors differ only on x_6 —thus x_6 is relevant, and is added to *NewRelevants*. Excluding x_6 from consideration too, we find that $\chi_{Base\cup\{x_3,x_7,x_8\}}$ is negative. This means that the number of relevant variables in $\{x_1, x_2, x_4\} \cup \{x_3, x_6, x_7, x_8\}$ is the same as the unknown threshold. So, we update *Base* from $\{x_1, x_2, x_4\}$ to $\{x_1, x_2, x_4, x_6\}$, and do BinarySearch from *Base* to $Base \cup \{x_3, x_7, x_8\}$. Ask MQ(11110110), the answer is 1. Ask MQ(11110100), the answer 1. Ask MQ(11010100), the answer is 0—thus x_3 is relevant. Testing $\chi_{\{x_1,x_2,x_3,x_4,x_6\}}$, we find that it is positive; thus since the number of relevant variables in $\{x_1, x_2, x_3, x_4, x_6, x_7, x_8\}$ is the same as the threshold, we know that $\{x_7, x_8\}$ contains no relevant variables.

A.2 Deleting the irrelevant variables

Now we know that $H = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ contains all the relevant variables; all that left is to get rid of the irrelevant ones (and determine the threshold).

This is done in FindDeletions. Procedure FindDeletions first determines a "big" positive and a "small" negative set. Suppose that we ask equivalence queries for TH_H^θ, for $\theta = 1, \ldots, |H|$. Since ψ is not constant, we must find two θ-values ℓ and u, and corresponding counterexamples χ_P and χ_N, such that $u = \ell + 1$, P is positive, and N is negative. Then it must also hold that $|P| \leq u - 1 = \ell \leq |N|$; thus N must contain an irrelevant element. In fact, we determine the above ℓ, u, P and N using binary search on the threshold value θ.

First, in Lines 1–6, we ask the two extreme cases $\mathrm{EQ}(\mathrm{TH}_H^{|H|})$ and $\mathrm{EQ}(\mathrm{TH}_H^1)$, getting counterexamples, say, 111110 and 000111.[3] The remainder of this binary search over threshold values is carried out in Lines 9–20. Ask $\mathrm{EQ}(\mathrm{TH}_H^4)$, the answer is NO, and suppose we receive the negative counterexample 001111. Ask $\mathrm{EQ}(\mathrm{TH}_U^5)$, the answer is NO, and suppose we receive the positive counterexample 111010. Now we have $u = 5$, $\ell = 4$, $P = \{x_1, x_2, x_3, x_5\}$ and $N = \{x_3, x_4, x_5, x_6\}$. Because P is already a minimal positive set, it does not change in the call to BinarySearch at Line 21.

Now, with the help of P, we determine an irrelevant variable of N as follows. We set their common part to be $Base = \{x_3, x_5\}$. The remaining parts of P and N, which are $P' = \{x_1, x_2\}$ and $N' = \{x_4, x_6\}$ are both even, so the call to MakeEven makes no changes (and returns *false*). We cut this remaining part of P' (resp. N') in two equal parts: $P_1 = \{x_1\}$ and $P_2 = \{x_2\}$ (resp. $N_1 = \{x_4\}$ and $N_2 = \{x_6\}$). Asking membership queries for all combinations $Base \cup P_i \cup N_j$, $i, j = 1, 2$, we find that $Base \cup P_1 \cup N_1$ is negative, meanwhile $Base \cup P_1 \cup P_2$ is positive. As P_2 has cardinality 1, this means that x_4 is irrelevant; remove it from H.

Now we restart, and conduct a binary search on the threshold value again, with the difference, that now $H = \{x_1, x_2, x_3, x_5, x_6\}$. Ask $\mathrm{EQ}(\mathrm{TH}_H^3)$, the answer is NO, and suppose we receive the negative counterexample 111000. Then asking $\mathrm{EQ}(\mathrm{TH}_H^4)$ the answer will be YES: our learning process has come to a successful end.

[3] As x_7 and x_8 are known to be irrelevant, from here on we shall omit the corresponding bits in the examples.