Journal of Machine Learning Research 3 (2002) 483-506 Submitted 6/02; Published 11/02

On Boosting with Polynomially Bounded Distributions

Nader H. Bshouty BSHOUTY@CS.TECHNION.AC.IL
Dmitry Gavinsky DEMITRY@CS.TECHNION.AC.IL
Department of Computer Science
Technion
Haifa, Israel, 32000

Editor: Philip M. Long

Abstract

We construct a framework which allows an algorithm to turn the distributions produced by some boosting algorithms into polynomially smooth distributions (w.r.t. the PAC oracle's distribution), with minimal performance loss.

Further, we explore the case of Freund and Schapire's *AdaBoost* algorithm, bounding its distributions to polynomially smooth. The main advantage of *AdaBoost* over other boosting techniques is that it is adaptive, i.e., it is able to take advantage of weak hypotheses that are more accurate than it was assumed a priori. We show that the feature of adaptiveness is preserved and improved by our technique.

Our scheme allows the execution of *AdaBoost* in the on-line boosting mode (i.e., to perform boosting "by filtering"). Executed this way (and possessing the quality of smoothness), now *AdaBoost* may be efficiently applied to a wider range of learning problems than before.

In particular, we demonstrate *AdaBoost*'s application to the task of *DNF learning using membership queries*. This application results in an algorithm that chooses the number of boosting iterations adaptively, and, consequently, adaptively chooses the size of the produced final hypothesis. This answers affirmatively a question posed by Jackson.

1. Introduction

Boosting, first introduced by Schapire (1990), is a learning method which demonstrates learning-theoretic equivalence between two learning models: the model of *distribution-free (strong) PAC-learning* and that of *distribution-free weak PAC-learning*. The PAC-model was first introduced by Valiant (1984), the strong and the weak cases were distinguished by Kearns and Valiant (1994).

The general framework for boosting is as follows. Suppose that we are dealing with some binary concept class, say C, which is a subclass of all functions from a domain X to $\{-1, 1\}$, and that there exists some concept $f \in C$ "known" to an oracle. Each time it is called, the oracle provides the learner with an instance $x \in X$, chosen according to some fixed but unknown to the learner *target distribution* D and with the corresponding labeling $f(x)$ of the instance. In the distribution-free (strong) PAC-learning model, the learner's aim is to produce, with probability at least $1 - \delta$, a *final hypothesis* $h(x)$ which $(1 - \varepsilon)$-approximates the concept c w.r.t. the distribution D, when ε and δ are parameters passed to the learner.

In the framework of boosting, the learner is supplied with an auxiliary algorithm WL (the *weak learner*) which, according to the distribution-free weak PAC-learning model, satisfies the following. Given an access to the standard PAC-oracle that answers according to some target function $f \in C$ over the distribution D_i, [1] WL produces a hypothesis h_i, which is a (possibly real-valued) mapping from X to $[-1, 1]$, s.t.

$$\frac{1}{2} - \gamma_i = \varepsilon_i \triangleq \mathop{\mathbf{E}}_{D_i}\left[\frac{|f(x) - h_i(x)|}{2}\right] \leq \frac{1}{2} - \gamma_{\min}$$

for some $\gamma_{\min}$ with confidence $1 - \delta_i$ (i.e., with probability at least $1 - \delta_i$). Note that $\gamma_{\min}$ is defined as a function of other parameters (n and δ) for a specific weak learner; in order for the learner to be efficient, $\gamma_{\min}$ should be inverse polynomial.

The booster is allowed to use WL as a subroutine, sequentially providing distributions D_i and receiving corresponding weak hypotheses h_i; afterwards it combines the received weak hypotheses in some manner producing the (potentially) strong PAC-hypothesis h_f. This "coupling" of booster and weak learner should possess all qualities required from a learner by the standard distribution-free PAC model and its complexity bounds should be polynomial in all the allowed complexity parameters.

One of the possibilities for choosing complexity parameters (which we adopt here) is as follows. We denote by n the value of $\log X$, or, in other words, the length of an instance representation (e.g., in Section 6 we will use $X = \{0, 1\}^n$). Similarly, by s we denote the (minimal) representation length of the target f. For both the strong and the weak PAC learning models, our set of complexity parameters includes n, s and the confidence parameter δ; for the strong model we also add ε to the set.

1.1 Smooth Boosting

Sometimes an additional restriction is put upon the booster: all the distributions D_i should be *polynomially smooth w.r.t. D* or *polynomially near-D*, i.e., for all i and all $x \in X$ it should hold that $D_i(x) \leq D(x) \cdot \alpha$, where α is the *smoothness parameter* and must be bounded above by a polynomial in (strong) PAC complexity parameters, as described above.

Among the known applications for this restriction are noise-tolerant learning (Freund, 1999, Domingo and Watanabe, 2000, Servedio, 2001), learning via extended statistical queries Bshouty and Feldman (2001), agnostic learning (Ben-David, Long and Mansour, 2001, Gavinsky, 2002), and learning concept classes using weak learning algorithms whose efficiency depends on the smoothness of the provided distribution D_i. Sometimes the smoothness restriction makes sense when the target distribution D is polynomially near-uniform, as it happens in the case of DNF membership learning (Jackson, 1997, Klivans and Servedio, 1999) considered in Section 6. Naturally, in this case all the (near-D) distributions D_i are polynomially near-uniform.

There exist a number of known smooth boosting algorithms (Freund, 1990, 1992, Klivans and Servedio, 1999, Freund, 1999, Domingo and Watanabe, 2000, Servedio, 2001), which show learning-theoretic equivalence between *poly-distribution-dependent strong PAC learning* and *poly-distribution-dependent weak PAC learning*. In other words, suppose that

1. Index i is used here for correspondence with further notation.

we have certain family of "favorable" distributions (over X), and we say that a distribution D is polynomially-close to the family if the infimum of the smoothness parameters of D w.r.t. the favorable family members is polynomially bounded. The following fact is implied by the existence of smooth boosting algorithms: Whenever a weak learner may be constructed which performs efficiently under distributions polynomially-close to favorable, a strong learner may be constructed whose performance would be efficient under those distributions. In particular, this holds for the case of (near-)uniform DNF learning.

1.2 Boosting Modes: Filtering versus Sampling

Boosting *by sampling* means that the learning is performed in two stages. In the first stage the algorithm collects a sufficient number of learning examples (i.e., a subset $S \subset X$) and in the second stage it performs the learning over this collection of examples only. The training collection S is polynomial in size.

After achieving a certain accuracy of the final hypothesis on the training set, information-theoretic techniques (based on Occam's Razor principle, VC dimension, etc.) may be applied to measure the overall accuracy (i.e., w.r.t. some distribution over X) of this hypothesis. The latter stage is sometimes called *generalization* and the (additional) error introduced by the domain widening is referred to as *generalization error*.

A possible reason for adopting this approach is that the booster is not smooth. A distribution which is not polynomially near-D cannot be efficiently simulated over a super-polynomially large domain.

On the other hand, the meaning of boosting *by filtering* is that the booster takes the whole set of instances as its learning domain. The examples received by the booster are not stored (the set S used by a sampling algorithm), but are "filtered": after being received from the PAC-oracle, an example is either forwarded to the current session of WL or "rejected" (i.e., not used at all).

This approach has two obvious advantages over boosting by sampling: the space complexity is reduced because of not storing the examples and no generalization error is introduced. At the same time, the analysis and the algorithm itself become slightly more complicated, as now the booster cannot get exact "statistics" by running through all the instances of the domain and needs to use some estimation schemes (based on Chernoff-like bounds).

As mentioned above, boosting by filtering may be used only if the underlying booster is polynomially smooth.

1.3 Our Results

A boosting algorithm which is not smooth may, on the other hand, possess some other "valuable properties"; we show that when certain assumptions regarding the algorithm's structure hold, it is possible to "force" the booster to produce only polynomially-smooth distributions. Moreover, our modification seems to be rather "neutral" towards other characteristics of the booster; in particular, in the specific case considered below we show that all the required properties of the booster are preserved when we apply our technique.

After introducing our technique, we will consider the case of the *AdaBoost* algorithm which was invented by Freund and Schapire (1997). This algorithm's main "valuable property" is *adaptiveness*; informally, this means that the algorithm is able to take advantage of weak hypotheses which are more accurate than it was assumed a priori A more formal definition of adaptiveness and further discussion may be found in Section 5.

Originally, the *AdaBoost* algorithm was designed so that different weak hypotheses "play different roles" in the final hypothesis: The final hypothesis is constructed in a form of weighted majority vote over weak hypotheses, and the weights are assigned in accordance with individual accuracies of the weak hypotheses; more accurate weak hypotheses receive more weight in the final construction. However, *AdaBoost* is (exponentially) not smooth (a proof may be found in Subsection 4.2).

Using our technique, we transform *AdaBoost* into a smooth booster, while the feature of adaptiveness is preserved (and in a sense improved, as we show in Section 5). In particular, we construct a modification of *AdaBoost*'s that works by filtering. Then we apply the modified *AdaBoost* to the task of learning DNF using membership queries, thus affirmatively answering the question posed by Jackson (1997).

In this connection, interesting work by Domingo and Watanabe (2000) should be mentioned. They construct a smooth modification of *AdaBoost* (which they call *MadaBoost*); however, their result is shown to be adaptive only for a decreasing sequence of weak hypotheses accuracies (i.e., when γ_i's declines as a function of i), which seems to be a quite strong limitation. Besides, their result applies only to the case of binary-valued weak hypotheses.

2. Definitions and Notation

We call a boosting algorithm producing only polynomially near-D distributions *polynomially near-D* (the word "polynomially" will be omitted sometimes).

Let B be a boosting algorithm learning some concept class C and using a weak learning algorithm WL. Consider the way B produces its final hypothesis from the weak hypotheses received during the boosting iterations: For example, the final hypothesis may be constructed as a weighted majority vote over all the received weak hypotheses (which is the case for *AdaBoost*). In such a final hypothesis structure, one interesting feature can be observed. Suppose that some weak hypothesis h_i which "appears" in the final hypothesis construction h_f is replaced by another hypothesis h'_i (locally, not affecting the other parts of the final hypothesis construction). Then, in the worst case, such change may increase the overall classification error of the final hypothesis by the value of

$$\varepsilon' \triangleq \Pr_{x \sim D}\left[h_i(x) = f(x), h'_i(x) \neq f(x)\right];$$

in particular, it holds that

$$\varepsilon' \leq \Pr_{x \sim D}\left[h'_i(x) \neq h_i(x)\right].$$

We call a final hypothesis structure possessing this quality and boosting algorithms producing such hypotheses *accuracy preserving*.

Throughout this essay we denote the mean value of a random variable V by $\mu[V]$. We denote the value of

$$\Pr_{x \sim D}[h(x) \neq f(x)]$$

by $\mathbf{Err}_D[h(x)]$, when the value of corresponding f is clear from the context.

3. Providing Near-D Bounds to Distributions

In this section we develop our technique for making the distributions generated by a boosting algorithm for a weak learner smooth.

Consider an accuracy preserving boosting algorithm B which collects T weak hypotheses before halting: If after the final hypothesis is constructed we add error of no more than ε' to each one of the T weak hypotheses, this may result, in the worst case, in $T \cdot \varepsilon'$ additional inaccuracy in the final hypothesis.

The modification we perform over B may be viewed as an additional layer constructed between the original booster and the weak learner. Naturally, the booster estimates some random values during its execution; in this introductory subsection, for simplicity and generality, we "assume" that we know exactly all the mean values that we need.

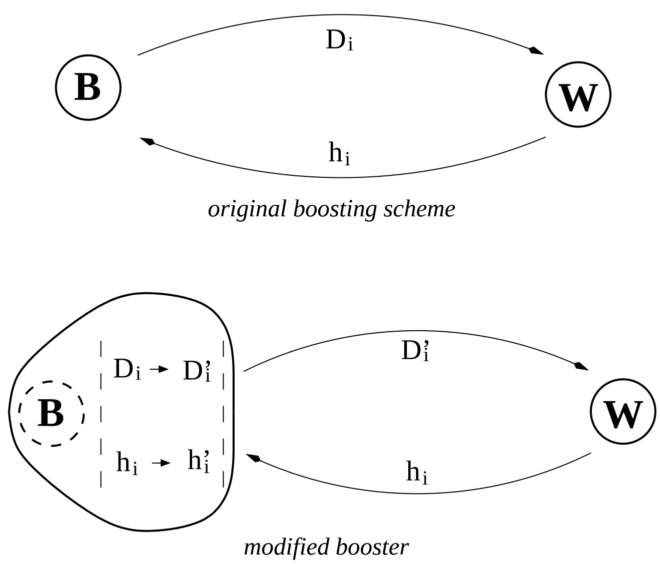

Figure 1: Modified boosting scheme

In order to perform our construction, we assume that for each distribution D_i supplied by B to the weak learner we can efficiently estimate the probability assigned by D_i to any instance $x \in X$. Below we show that this assumption is valid for *AdaBoost*.

Suppose that in the i'th stage the booster intends to provide a distribution D_i to the weak learner that is "too rough": This means that at least one instance $x_0 \in X$ has

$$D_i(x_0) > \eta \cdot D(x_0),$$

for some η to be fixed later. In this case the intermediate layer acts in the following manner. First it estimates the total weight w.r.t. D_i of all the instances $x \in X$ whose weight is above $\eta \cdot D(x)$; if this weight exceeds $\frac{3}{4}$, the weak learner is not called at all and an arbitrary hypothesis (e.g., const. -1) is returned to the booster. Otherwise, our intermediate layer "suppresses" all those instances whose weight exceeds $\eta \cdot D(x)$; i.e., those instances are not allowed to pass from the booster to the weak learner when the booster tries to send them. As a result of such "filtering", WL may, in general, be faced with a distribution different from D_i; we denote this new distribution by D'_i. For a chart describing our modification, see Figure 1.

According to the above principle, WL either was or was not executed during the i'th stage. It can be seen that if WL was executed then

$$D'_i(x) = \begin{cases} \frac{D_i(x)}{\Pr_{x\sim D_i}[D_i(x)\leq\eta\cdot D(x)]} & D_i(x) \leq \eta \cdot D(x) \\ 0 & otherwise \end{cases} . \tag{1}$$

Because in this case

$$\Pr_{x\sim D_i}[D_i(x) \leq \eta \cdot D(x)] \geq \frac{1}{4},$$

we have

$$D'_i(x) \leq \frac{D_i(x)}{\Pr_{x\sim D_i}[D_i(x) \leq \eta \cdot D(x)]} \leq \frac{\eta \cdot D(x)}{\Pr_{x\sim D_i}[D_i(x) \leq \eta \cdot D(x)]} \leq 4\eta \cdot D(x),$$

and the resulting distribution is therefore (4η)-near-D.

Therefore, the newly introduced layer successfully makes the general boosting scheme smooth, from WL's "point of view" (i.e., WL sees only smooth distributions). It remains to check what happens on B's side – whether the received weak hypotheses may be used in order to construct a successful final hypothesis. In general, the weak hypotheses returned to the booster do not meet the requirements of the weak learner. For example, this can be easily observed in a case when a constant -1 hypothesis is returned to the booster.

In order to improve the situation, we "force" the boosting algorithm to behave as if all the "high weight instances" of the distribution D_i (i.e., such $x \in X$ that $D_i(x) > \eta \cdot D(x)$) were classified correctly by the weak hypothesis received. More specifically, if at the i'th stage our modified booster receives some hypothesis h_i then it should behave as the original algorithm would upon receiving of

$$h'_i(x) = \begin{cases} h_i(x) & if\ D_i(x) \leq \eta \cdot D(x) \\ f(x) & otherwise \end{cases} , \tag{2}$$

when $f(x)$ denotes the target concept, as defined before. Of course, not all boosting algorithms allow such modifications; we assume that our B does allow them. We show below that this is the case when *AdaBoost* is playing B's role.

Let us examine the consequences of the above modification: First, we have to verify that $h'_i(x)$ is always at least $(1/2 + \gamma)$-accurate w.r.t. D_i; second, we have to review the influence of the modification upon the final hypothesis' accuracy (which results from the fact that the final hypothesis will be constructed out of the "actual" h_i's instead of the "virtual" h'_i's).

For the first check, note that, if during the i'th iteration WL was not executed by the intermediate layer, then $\Pr_{x\sim D_i}[D_i(x) > \eta \cdot D(x)] \geq \frac{3}{4}$ and the accuracy of $h'_i(x)$ w.r.t. D_i is as well above $3/4$, which is good enough.[2] Otherwise (when WL was executed) note that the received weak hypothesis may be assumed to be $(\frac{1}{2} + \gamma)$-accurate w.r.t. D'_i, in accordance with WL's specification. Therefore, as follows from (1) and (2), in this case

2. Formally assume, that before the algorithm started, we had set $\gamma \rightarrow \min\{\gamma, 1/4\}$.

$h'_i(x)$ is accurate enough as well:

$$\begin{aligned}
&\mu_{x\sim D_i}\left[\tfrac{|h'_i(x)-f(x)|}{2}\,\middle|\, D_i(x) \le \eta\cdot D(x)\right] \\
&\quad = \mu_{x\sim D_i}\left[\tfrac{|h_i(x)-f(x)|}{2}\,\middle|\, D_i(x) \le \eta\cdot D(x)\right] \\
&\quad = \mu_{x\sim D'_i}\left[\tfrac{|h_i(x)-f(x)|}{2}\right] \\
&\quad \le \tfrac{1}{2}-\gamma, \\
&\mu_{x\sim D_i}\left[\tfrac{|h'_i(x)-f(x)|}{2}\,\middle|\, D_i(x) > \eta\cdot D(x)\right] = 0,
\end{aligned}$$

and therefore

$$\mu_{x\sim D_i}\left[\frac{|h'_i(x)-f(x)|}{2}\right] \le \frac{1}{2}-\gamma.$$

So, we see that the h'_i's are always at least $(1/2+\gamma)$-accurate w.r.t. D_i.

For the second check, we are going to make use of the fact that B's final hypothesis is accuracy-preserving. Let us estimate the maximum error which may be added by using h_i instead of h'_i (for a single i)

$$\begin{aligned}
\mu_{x\sim D}\left[\tfrac{|h_i(x)-f(x)|}{2}\right] - \mu_{x\sim D}\left[\tfrac{|h'_i(x)-f(x)|}{2}\right] \quad &\le \Pr_{x\sim D}\left[h'_i \ne h_i\right] \\
&\le \Pr_{x\sim D}\left[D_i(x) > \eta\cdot D(x)\right] \\
&< \tfrac{1}{\eta} \triangleq \varepsilon'.
\end{aligned}$$

If B performs T iterations then the maximum additional error of the final hypothesis h_f which may be added by application of our smoothing technique is $\frac{T}{\eta}$.

So, we have developed a technique whose analysis gives rise to the following claim.

Proposition 1 *Assume that the structure of an accuracy preserving boosting algorithm B allows the described modifications; suppose also that we know that B performs not more than T boosting iterations in order to produce a final hypothesis whose error would be bounded above by ε.*

Then it is possible to modify the boosting algorithm so that by performing the same number of iterations, but providing only η-smooth distributions to WL, it will produce a final hypothesis whose error would be not higher than $\varepsilon + \frac{T}{\eta}$.

The confidence parameter δ which was not explicitly dealt with in this section does not affect the result in any significant way (we have supposed that all the calls to WL were successful).

4. The Case of *AdaBoost*

As mentioned above (and as follows from its name), *AdaBoost* is adaptive but it is not smooth; we would like to turn it into a smooth boosting algorithm.

First note that the final hypothesis produced by *AdaBoost* is accuracy preserving. In order to be able to apply our technique, we need some upper bound T_{upper} on the number of iterations performed – otherwise we cannot decide which value may be chosen for the parameter η, as required by Proposition 1. However, because the booster is adaptive, the number of iterations depends on all the weak hypotheses received during the boosting iterations; on the other hand, if we fix T a priori, based on the least possible value for γ_i's, then we make the algorithm non-adaptive.

This apparent contradiction may be resolved in the following way: We do not fix the actual number of the boosting iterations to be performed, and it remains adaptive; however, we make an a priori "worst case" estimation for T and we use this estimation in order to appropriately fix a value for η. Denote the least possible value for any of the γ_i's by $\gamma_{\min}$; as shown by Freund and Schapire (1997), in this case the number T of *AdaBoost*'s iterations satisfies

$$T \leq \left\lceil \frac{1}{2\gamma_{\min}^2} \ln \frac{1}{\varepsilon} \right\rceil .$$

4.1 Boosting by Filtering

In this section we apply our technique to construct a smooth modification of *AdaBoost* that performs boosting by filtering.

We no longer assume that WL is "confident", now it is executed with a confidence parameter δ' (the allowed probability to fail).

We call our algorithm *QuickFilt*; it is represented in Figures 2 and 3. The parameters passed to this algorithm are $(EX, WL, \gamma_{\min}, \varepsilon, \delta)$, where EX is a PAC oracle.

In accordance with the principles defined above, in our algorithm we denote by T_{upper} our a priori upper bound on the number of iterations. We find a satisfactory value for T_{upper} using the supplied parameter $\gamma_{\min}$; if this value is higher than some constant, we redefine $\gamma_{\min}$ to that constant value.

The original *AdaBoost* algorithm always holds a "weight" corresponding to each one of the sample's instances and updates these values between iterations: During a given boosting iteration, the weight $w(x_0)$ corresponding to some instance x_0 defines (being multiplied by a normalizing factor) the probability of the pair $(x_0, f(x_0))$ to be given as an example to the weak learner. In our case, however, we cannot permanently maintain all those weights (simply since the learning domain is "too large"); instead, we use a subroutine *Get_w* (described below) which calculates the current value of $w(x_0)$ for a specific x_0.

4.1.1 *QuickFilt*'s Analysis

The *QuickFilt* algorithm has a subroutine *Evaluate* which is used to estimate mean values of random variables, based on the Hoeffding bound.[3] Called with parameters $(X, \frac{\lambda}{b-a}, \delta)$, this subroutine estimates the mean value of X with error bounded from above by λ and with confidence parameter δ (which is the allowed probability that the estimation fails). Its time and sample complexity is

$$O\left(\frac{\ln\left(\frac{1}{\delta}\right)}{\left(\frac{\lambda}{b-a}\right)^2} \right).$$

3. The Hoeffding bound is $\Pr\left[\left|\frac{1}{m}\sum_{i=1}^{m} x_i - \mu\right| \geq \lambda\right] \leq 2\exp\left(-2\left(\frac{\lambda}{b-a}\right)^2 \cdot m\right)$, where x_i's are independent samples from a random variable X with mean value μ and values from the real range $[a, b]$.

QuickFilt$(EX, WL, \gamma_{\min}, \varepsilon, \delta)$

1. **set:** $\gamma'_{\min} = \min\{\gamma_{\min}, \frac{1}{30}\}$; $T_{upper} = \left\lceil \frac{1}{2\gamma'^2_{\min}} \ln\left(\frac{6}{\varepsilon}\right) \right\rceil$
2. **set:** $\varepsilon_c = 1$; $i = 1$
3. **while** $(\varepsilon_c \geq \frac{5\varepsilon}{6})$
4. **let:** X be a variable drawn $\sim D$;
   let: Y be a correlated variable $= Get\_w(i, X, f(X))$
5. **set:** $Z_i = Evaluate(Y, \frac{\varepsilon}{54T_{upper}}, \frac{\delta}{4T_{upper}})$
6. **call:** *WL*$(\delta' = \frac{\delta}{4T_{upper}})$*, providing it with distribution generated by* $D_i gen$*; denote the returned weak hypothesis by* h_i
7. **define:** $h'_i(x) \triangleq \begin{cases} h_i(x) & if\ w(x) \leq \frac{6T_{upper}Z_i}{\varepsilon} \\ f(x) & otherwise \end{cases}$
8. **let:** X be a variable drawn $\sim D_i gen$;
   let: Y be a correlated variable $= \frac{|h'_i(X) - f(X)|}{2}$
9. **set:** $\varepsilon_i = Evaluate(Y, \frac{\varepsilon}{64T_{upper}}, \frac{\delta}{4T_{upper}})$
10. **set:** $\beta_i = \max\left\{\frac{\varepsilon_i}{1-\varepsilon_i}, \frac{1}{2}\right\}$
11. **define:** $h_{prev}(x) = \begin{cases} 1 & if\ \sum_{t=1}^{i}(\log \frac{1}{\beta_t}) h_t(x) \geq 0 \\ -1 & otherwise \end{cases}$
12. **let:** X be a variable drawn $\sim D$;
    let: Y be a correlated variable $= \frac{|h_{prev}(X) - f(X)|}{2}$
13. **set:** $\varepsilon_c = Evaluate(Y, \frac{\varepsilon}{6}, \frac{\delta}{4T_{upper}})$
14. **set:** $i = i + 1$
15. **end-while**
16. *Output the final hypothesis:* $h_f(x) = h_{prev}(x)$

Figure 2: The *QuickFilt*$(EX, WL, \gamma_{\min}, \varepsilon, \delta)$ hypothesis boosting algorithm.

In the case of filtering, we cannot permanently keep the value of $w(x)$ for each instance. Subroutine *Get_w* is used to calculate $w^i(x)$'s (which is the value of $w(x)$ in the i'th iteration); sometimes we will denote for brevity the output produced by $Get\_w(i, x, f(x))$ simply by $w^i(x)$.

For *QuickFilt*'s analysis, we use the notation of $\langle\ \overline{value}\ \rangle$ to denote "true" values, as distinct from estimated values produced by the algorithm. For example, we denote

$$\overline{\varepsilon_i} = \mu_{x \sim D_i}\left[\frac{|h'_i(x) - f(x)|}{2}\right]$$

and ε_i, as defined by *QuickFilt*, is a result of the corresponding estimation; similarly, by β_i we denote the value that is assigned in line 10 to the variable of the same name during a specific algorithm's execution, while $\overline{\beta_i}$ is the value that would be received using $\overline{\varepsilon_i}$ instead of ε_i: $\overline{\beta_i} = \max\left\{\frac{\overline{\varepsilon_i}}{1-\overline{\varepsilon_i}}, \frac{1}{2}\right\}$. Further, we denote the corresponding estimation inaccuracy by

$Evaluate(X, \frac{\lambda}{b-a}, \delta)$
  set: $m = \left\lceil \ln\left(\frac{2}{\delta}\right) / 2\left(\frac{\lambda}{b-a}\right)^2 \right\rceil$
  return $\frac{\sum_{i=1}^{m} X_i}{m}$

$D_i gen$
  do:
    receive: $(x, f(x))$ from EX; **draw randomly:** $r \sim \mathbf{U}_{0,1}$
    if $(r < Get\_w(i, x, f(x)) \cdot \frac{\varepsilon}{27 T_{upper} Z_i})$ **then return** $(x, f(x))$
  end-do

$Get\_w(i, x, f(x))$
  set: $w' = 1, k = 1$
  while $(k < i)$
    set: $w' = w' \cdot \beta_k^{1 - \frac{|h'_k(x) - f(x)|}{2}}$
    set: $k = k + 1$
  end-while
  return $w^i(x) = w'$

Figure 3: Subroutines used in *QuickFilt* hypothesis boosting algorithm.

$\mathbf{err}\,[< value >]$, i.e.,

$$\mathbf{err}\,[< value >] = \left| < value > - \left\langle\ \overline{value}\ \right\rangle \right|.$$

The subroutine $D_i gen$ is used to produce distributions for the weak learner. The distribution produced by $D_i gen$ during the i'th iteration is denoted by D_i (with an exception of line 9 of *QuickFilt*, where a more "constructive" notation is used).

Claim 2 *Each time lines* 6-10 *of QuickFilt are executed, with probability at least* $1 - \frac{\delta}{2T_{upper}}$ *the following statements are true:*

- *The resulting weak hypothesis* h'_i *satisfies w.r.t.* D_i*:*

$$\frac{1}{2} - \gamma_{\min} \geq \overline{\varepsilon_i}, \;\; \mathbf{err}\,[\varepsilon_i] \leq \frac{\varepsilon}{64 T_{upper}}, \;\; \mathbf{err}\,[\beta_i] \leq \frac{\varepsilon}{16 T_{upper}}.$$

- *It holds that*

$$\frac{1}{2} \leq \beta_i < 1.$$

Proof of Claim 2 Follows from the estimations performed by *QuickFilt*.
$\blacksquare_{\textit{Claim 2}}$

Note that in *QuickFilt* no distribution "smoothing" is actually performed: the required smoothness of D_i results from the h'_{i-1}'s accuracy over the "high" instances of D_{i-1}. We show that all the distributions produced by *QuickFilt* are polynomially near uniform.

Claim 3 *Let T be the overall number of boosting iterations performed, then with probability at least $1 - \frac{T\delta}{4T_{upper}}$ the following statements hold:*

- *All the estimations of $\overline{Z_i}$ performed in line 5 of QuickFilt are accurate up to the multiplicative factor $\frac{3}{2}$, i.e., $\frac{2}{3}\overline{Z_i} \leq Z_i \leq \frac{3}{2}\overline{Z_i}$.*
- *For each i, $x \in X$,*

$$D_i(x) = \frac{w^i(x)D(x)}{\sum_{x \in X} w^i(x)D(x)} = \frac{w^i(x)}{\overline{Z_i}} \cdot D(x). \tag{3}$$

- *All the distributions D_i provided to WL during boosting are $(\frac{18T_{upper}}{\varepsilon})$-near D; i.e., for each i, $x \in X$,*

$$D_i(x) \leq \frac{18T_{upper}}{\varepsilon} \cdot D(x). \tag{4}$$

Proof of Claim 3 With probability at least $1 - \frac{T\delta}{4T_{upper}}$ all the estimations of Z_i performed in line 5 of *QuickFilt* are accurate within the allowed value for $\frac{\lambda}{b-a}$, i.e., $\mathbf{err}\left[Z_i\right] \leq \frac{\varepsilon}{54T_{upper}}$ $(0 < Z_i \leq 1)$; from this point on we assume that this is the case.

The further proof possesses the structure of induction. For the basis notice that at the first boosting stage ($i = 1$) the claim holds.

Suppose that the claim is true for some i; in particular, it holds that

$$\forall x: \ D_i(x) = \frac{w^i(x)}{\overline{Z_i}} \cdot D(x).$$

Consider the way the weights are updated between iterations (see *Get_w* subroutine): each weight either remains the same or is multiplied by β_i. Because $\frac{1}{2} \leq \beta_i < 1$ (Claim 2), it holds that $\overline{Z_{i+1}} \geq \beta_i \cdot \overline{Z_i} \geq \frac{1}{2}\overline{Z_i}$. Also note that during the estimation of Z_{i+1}, the given value of $\frac{\lambda}{b-a}$ satisfies

$$\frac{\lambda}{b-a} = \frac{\varepsilon}{54T_{upper}} \ . \tag{5}$$

As follows from h'_i's definition (line 7 of *QuickFilt*), during the i'th iteration the weights of all the instances $x \in X$ satisfy

$$\left(\frac{18T_{upper}\overline{Z_i}}{\varepsilon} \geq\right) \ w^i(x) > \frac{9T_{upper}\overline{Z_i}}{\varepsilon} \geq \frac{6T_{upper}Z_i}{\varepsilon}$$

are correctly classified by h'_i's and thus multiplied by β_i; therefore the probabilities of corresponding instances do not rise before the next iteration. Among x's satisfying $w^i(x) \leq 6T_{upper}Z_i/\varepsilon$, the greatest individual probabilities (and $w(x)$'s) for the next iteration can be achieved for instances x where

$$w^i(x) = \frac{9T_{upper}\overline{Z_i}}{\varepsilon}$$

(which is possible when $Z_i = \frac{3}{2}\overline{Z_i}$); the corresponding $w(x)$'s would satisfy

$$\frac{w^{i+1}(x)}{\overline{Z_{i+1}}} = \frac{w^i(x)}{\overline{Z_{i+1}}} \leq 2\frac{w^i(x)}{\overline{Z_i}} \leq \frac{18T_{upper}}{\varepsilon}.$$

Consequently, for each $x \in X$ it holds that

$$w^{i+1}(x) \leq \frac{18T_{upper}\overline{Z_{i+1}}\varepsilon}{}. \tag{6}$$

From Equations (5) and (6), for the estimation of Z_{i+1} (line 5 of *QuickFilt*),

$$\frac{\lambda}{b-a} \leq \frac{\overline{Z_{i+1}}/3}{w^{i+1}_{\max}}$$

holds, where $w^{i+1}_{\max} \triangleq \max\{w^{i+1}(x)\}$. Since in this case $b - a \leq w^{i+1}_{\max}$, therefore

$$\lambda \leq \overline{Z_{i+1}}/3.$$

This fulfills the proof of Z estimations accuracy.

It follows from (6) that (3) and (4) hold. This fulfills the induction step.

The result follows.

$\blacksquare$ *Claim 3*

It was shown by Jackson (1997) that for the boosting algorithm used there it is possible to estimate efficiently, up to a constant multiplicative factor, the distributions D_i provided to the weak learner (by distribution estimation we mean the ability to evaluate the probabilistic weight assigned to each individual point in the domain). Similarly, such estimation is possible for *QuickFilt*.

This feature is critical for the possibility of application of the boosting algorithm to the task of DNF-learning with membership queries under uniform distributions.

Theorem 4 *Suppose QuickFilt was executed with arguments (EX, WL, $\gamma_{\min}$, ε, δ) and that each time WL was called with argument δ' over distribution D_i, it produced a hypothesis h_i s.t.*

$$\Pr\left[\mu_{x\sim D_i}\left[\frac{|h_i(x) - f(x)|}{2}\right] \leq \frac{1}{2} - \gamma_{\min}\right] \geq 1 - \delta'.$$

Then with probability at least $(1 - \delta)$, the algorithm QuickFilt halts and returns a final hypothesis which makes $(1 - \varepsilon)$-correct predictions over the instance space.

Proof of Theorem 4 Suppose that *QuickFilt* performed T iterations and WL returned hypotheses $h_1, ..., h_T$.

With probability at least $1-\frac{T\delta}{4T_{upper}}$, the ε_c-estimation performed in line 13 of $QuickFilt$ always satisfies the accuracy requirement. The overall probability for this assumption together with all the assumptions previously made (in the proofs of Claims 2 and 3) to hold is at least $1-\frac{T\delta}{T_{upper}}$.

As follows from Claim 3 and from the construction of Get_w, the distributions D_i's produced by $D_i gen$ subroutine are identical to those defined "according to the principle of $AdaBoost$", i.e.,

$$D_i(x)=\frac{D(x)\cdot\prod_{k=1}^{i-1}\beta_k^{1-\frac{|h'_k(x)-f(x)|}{2}}}{\sum_{x\in X}\left[\prod_{k=1}^{i-1}\beta_k^{1-\frac{|h'_k(x)-f(x)|}{2}}\cdot D(x)\right]}=\frac{w^i(x)\cdot D(x)}{\sum_{x\in X}[w^i(x)\cdot D(x)]}\ .$$

Recall that $\frac{1}{2}>\overline{\varepsilon_i}$ and $\max\{\frac{1}{2},\frac{\varepsilon_i}{1-\varepsilon_i}\}=\beta_i<1$. Consequently, if in i'th iteration it holds that $\beta_i=\frac{\varepsilon_i}{1-\varepsilon_i}$, then

$$\begin{array}{ll}\sum_{x\in X}w^{i+1}(x)D(x) & =\sum_{x\in X}w^i(x)D(x)\beta_i^{1-\frac{|h'_i(x)-f(x)|}{2}}\\ & \leq\sum_{x\in X}w^i(x)D(x)(1-(1-\beta_i)(1-\frac{|h'_i(x)-f(x)|}{2}))\\ & =\left(\sum_{x\in X}w^i(x)D(x)\right)(1-(1-\overline{\varepsilon_i})(1-\beta_i))\\ & \leq\left(\sum_{x\in X}w^i(x)D(x)\right)(1-(1-\overline{\varepsilon_i})(1-\overline{\beta_i}-\frac{\varepsilon}{16T_{upper}}))\\ & <\left(\sum_{x\in X}w^i(x)D(x)\right)\cdot(2\overline{\varepsilon_i}+\frac{\varepsilon}{16T_{upper}})\ .\end{array}$$

Otherwise $\beta_i=\frac{1}{2}$, $\varepsilon_i<\frac{1}{3}$, $\overline{\varepsilon_i}<\frac{1}{3}+\frac{\varepsilon}{64T_{upper}}$ and

$$\begin{array}{ll}\sum_{x\in X}w^{i+1}(x)D(x) & \leq\left(\sum_{x\in X}w^i(x)D(x)\right)(1-(1-\overline{\varepsilon_i})(1-\beta_i))\\ & =\left(\sum_{x\in X}w^i(x)D(x)\right)(1-\frac{1}{2}(1-\overline{\varepsilon_i}))\\ & =\left(\sum_{x\in X}w^i(x)D(x)\right)(\frac{2}{3}+\frac{\varepsilon}{128T_{upper}})\ .\end{array}$$

Therefore

$$\sum_{x\in X}w^{T+1}(x)D(x)\leq\prod_{i=1}^{T}\left(2\max\{\overline{\varepsilon_i},1/3\}+\frac{\varepsilon}{16T_{upper}}\right). \tag{7}$$

Let us now define a "virtual" version of the final hypothesis:

$$\widehat{h_{prev}}(x)=\left\{\begin{array}{ll}1 & if\ \sum_{t=1}^{T}(\log\frac{1}{\beta_t})h'_t(x)\geq 0\\ -1 & otherwise\end{array}\right.,$$

and denote $\varepsilon'_f\triangleq\mu_{x\sim D}\left[\frac{|\widehat{h_{prev}}(x)-f(x)|}{2}\right]$. Then $\widehat{h_{prev}}(x)$ makes a mistake on x only if

$$\prod_{i=1}^{T}\beta_i^{-\frac{|h'_i(x)-f(x)|}{2}}\geq\left(\prod_{i=1}^{T}\beta_i\right)^{-1/2}.$$

Since $w^{T+1}(x) = \prod_{i=1}^{T} \beta_i^{1-\frac{|h'_i(x)-f(x)|}{2}}$, it holds that

$$\sum_{x \in X} w^{T+1}(x) D(x) \geq \varepsilon'_f \cdot \left(\prod_{i=1}^{T} \beta_i \right)^{1/2} .$$

From (7) it follows that

$$\begin{aligned} \varepsilon'_f & \leq \prod_{i=1}^{T} \frac{2 \max\{\overline{\varepsilon_i}, 1/3\} + \frac{\varepsilon}{16 T_{upper}}}{\sqrt{\beta_i}} \\ & \leq \prod_{i=1}^{T} \max \left\{ \frac{2\overline{\varepsilon_i} + \frac{\varepsilon}{16 T_{upper}}}{\sqrt{\frac{\overline{\varepsilon_i}}{1-\overline{\varepsilon_i}} - \frac{\varepsilon}{16 T_{upper}}}}, \frac{\frac{2}{3} + \frac{\varepsilon}{16 T_{upper}}}{\frac{1}{\sqrt{2}}} \right\}, \end{aligned}$$

and because $\frac{\varepsilon}{64 T_{upper}} \leq \min\{\gamma'^2_{\min}, \frac{1}{16}\}$ and $\frac{1}{2} - \gamma'_{\min} \geq \overline{\varepsilon_i}$, we get

$$\varepsilon'_f \leq \prod_{i=1}^{T} 2 \max \left\{ \sqrt{\overline{\varepsilon_i}(1-\overline{\varepsilon_i})}, \sqrt{2} \middle/ 3 \right\} + T \frac{\varepsilon}{4 T_{upper}}.$$

Denote: $\gamma_i^{\min} \triangleq \min\left\{\frac{1}{2} - \overline{\varepsilon_i}, \frac{1}{6}\right\}$, then

$$\varepsilon'_f \leq \prod_{i=1}^{T} \sqrt{1 - 4\gamma_i^{\min 2}} + T \frac{\varepsilon}{4 T_{upper}}.$$

Once again we use the transformation from Freund and Schapire (1997), thus achieving

$$\begin{aligned} \varepsilon'_f & \leq \exp\left(-2 \sum_{i=1}^{T} \gamma_i^{\min 2}\right) + T \frac{\varepsilon}{4 T_{upper}} \\ & \leq \exp(-2T \cdot \gamma'^2_{\min}) + T \frac{\varepsilon}{4 T_{upper}}. \end{aligned} \tag{8}$$

In order to bound the overall error, it remains to estimate the "differences" between the functions h_i used to produce the final hypothesis and corresponding "virtual" functions h'_i considered in the analysis above.

As follows from the construction of the algorithm,

$$\varepsilon' \triangleq \mu\left[\frac{|h_i(x) - h'_i(x)|}{2}\right] \geq \mu\left[\frac{|h_i(x) - f(x)|}{2}\right] - \mu\left[\frac{|h'_i(x) - f(x)|}{2}\right],$$

and the (actual) final error $\overline{\varepsilon_c}$ satisfies that $\overline{\varepsilon_c} \leq \varepsilon'_f + T \cdot \varepsilon'$. From the h'_i's definition, $h_i(x) \neq h'_i(x)$ only for those instances that satisfy $w(x) > \frac{4 T_{upper} \overline{Z_i}}{\varepsilon}$, and therefore $\varepsilon' \leq \frac{\varepsilon}{4 T_{upper}}$. Now it can be inferred that

$$\overline{\varepsilon_c} \leq \varepsilon'_f + \frac{T\varepsilon}{4 T_{upper}}. \tag{9}$$

We have assumed that the ε_c-estimation performed in line 13 of $QuickFilt$ keeps within the accuracy required; therefore the algorithm halts as soon as $\varepsilon_c \leq \overline{\varepsilon_c} + \frac{\varepsilon}{6} \leq \frac{5\varepsilon}{6}$, i.e., when

$\overline{\varepsilon_c} \leq \frac{2\varepsilon}{3}$. By substitution of this expression into (9), we conclude that a (sufficient) halt condition is $\varepsilon'_f + \frac{T\varepsilon}{4T_{upper}} \leq \frac{2\varepsilon}{3}$, and using (8) we may replace it by

$$\exp(-2T \cdot \gamma'^2_{\min}) + T\left(\frac{\varepsilon}{4T_{upper}} + \frac{\varepsilon}{4T_{upper}}\right) \leq \frac{2\varepsilon}{3}.$$

Suppose that

$$T\left(\frac{\varepsilon}{4T_{upper}} + \frac{\varepsilon}{4T_{upper}}\right) \leq \frac{\varepsilon}{2}\ , \tag{10}$$

then the algorithm halts when

$$\exp(2T\gamma'^2_{\min}) \geq \frac{6}{\varepsilon}, \tag{11}$$

which is satisfied by $T = T_{upper}$, as defined in *QuickFilt*. Also, assumption (10) holds for $T = T_{upper}$ and therefore the algorithm halts after T_{upper} iterations at most.

As mentioned above, the probability for all the assumptions made in the beginning of the proof to hold is at least $1-\delta$, which completes the proof.

$\blacksquare$ *Theorem 4*

Note that sometimes WL's complexity depends upon the smoothness of the provided distribution. In the present analysis we leave the issue aside, it will be considered in Section 6, dealing with DNF learning.

Theorem 5 *Suppose that all the assumptions of Theorem 4 hold. Suppose also that if WL was called in the i'th iteration, provided with distribution D_i, it returned a weak hypothesis h_i satisfying $\mu_{x\sim D_i}\left[\frac{|h_i(x)-f(x)|}{2}\right] = \frac{1}{2} - \gamma_i$. Then QuickFilt($EX$, WL, $\gamma_{\min}$, ε, δ) performs*

$$O\left(\frac{\ln(\varepsilon^{-1})}{\mu_{1\leq i\leq T}\left[\gamma_i^2\right]}\right)$$

iterations, makes

$$\tilde{O}\left(\frac{Q[WL]}{\varepsilon\gamma^2_{\min} \cdot \mu_{1\leq i\leq T}\left[\gamma_i^2\right]} + \frac{1}{\varepsilon^3\gamma^6_{\min} \cdot \mu_{1\leq i\leq T}\left[\gamma_i^2\right]}\right)$$

queries and halts in time

$$\tilde{O}\left(\frac{T[WL]}{\mu_{1\leq i\leq T}\left[\gamma_i^2\right]} + \frac{Q[WL]}{\varepsilon\gamma^2_{\min}\left(\mu_{1\leq i\leq T}\left[\gamma_i^2\right]\right)^2} + \frac{1}{\varepsilon^3\gamma^6_{\min}\left(\mu_{1\leq i\leq T}\left[\gamma_i^2\right]\right)^2}\right).$$

The final hypothesis is represented as a weighted majority vote of the weak hypotheses whose number equals the number of iterations performed, and all the distributions provided to WL are $O\left(\frac{\ln(\varepsilon^{-1})}{\varepsilon\gamma^2_{\min}}\right)$-near D.

Proof of Theorem 5 First note that the statements of the last sentence of the theorem are obviously true if the rest of the theorem statements hold.

As shown in the proof of Theorem 4, the algorithm always halts within $T \le T_{upper}$ iterations. Denote $T_1 \triangleq \left\{ i \,\middle|\, \beta_i = \frac{\varepsilon_i}{1-\varepsilon_i} \right\}$, $T_2 \triangleq \{1, ..., T\} \setminus T_1$.

By analogy to Equation (11), a sufficient condition for halting is that

$$\exp\left(2|T_1| \cdot \underset{i \in T_1}{\mu} \left[\gamma_i^2 \right] + 2|T_2| \cdot \left(\frac{1}{30} \right)^2 \right) \ge \frac{6}{\varepsilon}.$$

Consequently,

$$T_1 \le \left\lceil \frac{1}{2\,\mu_{i \in T_1} \left[\gamma_i^2 \right]} \ln\left(\frac{6}{\varepsilon} \right) \right\rceil, \; T_2 \le \left\lceil 450 \ln\left(\frac{6}{\varepsilon} \right) \right\rceil.$$

Note that a subscript of "$i \in T_1$" appears in the last expressions. In order to prove our theorem we would like to change it into "$1 \le i \le T$". The former range for i corresponds to the latter one with exclusion of the indices of iterations when "actual" β_i was replaced by 1/2. But this does not really harm the algorithm's performance, if we evaluate the complexity in terms of O:

$$\underset{i \in T_1}{\mu} \left[\gamma_i^2 \right] = \frac{1}{|T_1|} \sum_{i \in T_1} \gamma_i^2 \ge \frac{1}{|T_1|} \left(\sum_{1 \le i \le T} \gamma_i^2 - \frac{1}{4} |T_2| \right) \ge \underset{1 \le i \le T}{\mu} \left[\gamma_i^2 \right] - \frac{|T_2|}{4|T_1|}$$

and consequently,

$$T_1 \le \left\lceil \frac{1}{2\,\mu_{1 \le i \le T} \left[\gamma_i^2 \right] - \frac{|T_2|}{2|T_1|}} \ln\left(\frac{6}{\varepsilon} \right) \right\rceil.$$

Further, we consider two cases:

1.

$$\frac{|T_2|}{2|T_1|} \le \underset{1 \le i \le T}{\mu} \left[\gamma_i^2 \right] \rightarrow T_1 \le \left\lceil \frac{1}{\mu_{1 \le i \le T} \left[\gamma_i^2 \right]} \ln\left(\frac{6}{\varepsilon} \right) \right\rceil = O\left(\frac{\ln(\varepsilon^{-1})}{\mu_{1 \le i \le T} \left[\gamma_i^2 \right]} \right).$$

2.

$$\frac{|T_2|}{2|T_1|} > \underset{1 \le i \le T}{\mu} \left[\gamma_i^2 \right] \rightarrow T_1 < \frac{|T_2|}{2\,\mu_{1 \le i \le T} \left[\gamma_i^2 \right]} = O\left(\frac{\ln(\varepsilon^{-1})}{\mu_{1 \le i \le T} \left[\gamma_i^2 \right]} \right).$$

In any case, it holds that

$$T = T_1 + T_2 = O\left(\frac{\ln(\varepsilon^{-1})}{\mu_{1 \le i \le T} \left[\gamma_i^2 \right]} \right). \tag{12}$$

We estimate the overall time complexity of the algorithm in two stages: first we deal with complexity of the whole algorithm apart from the $D_i gen$'s executions, and then we consider the overall complexity of $D_i gen$.

Each one of *Evaluate*'s occurrences is encountered at most once during each iteration, but some calls to *Evaluate* provide it with argument X which depends on $w(x)$. Because the time complexity of *Get_w* is $O(T)$ and $\left(\frac{\lambda}{b-a}\right)^{-1} = O\left(\varepsilon / T_{upper}\right)$, a single *Evaluate* call may consume up to

$$O\left(\ln(\frac{T_{upper}}{\delta})\frac{T \cdot T_{upper}^2}{\varepsilon^2}\right) = \tilde{O}\left(\frac{T \cdot T_{upper}^2}{\varepsilon^2}\right) = \tilde{O}\left(\frac{T}{\gamma_{\min}^4 \varepsilon^2}\right)$$

time. The time complexity of the whole algorithm apart from $D_i gen$ is bounded by

$$\tilde{O}\left(T \cdot \left(\frac{T}{\gamma_{\min}^4 \varepsilon^2} + T[WL]\right)\right),$$

assuming that $T[WL(\gamma_{WL})]$ is logarithmically bounded w.r.t. its confidence parameter γ_{WL}.

It remains to estimate $D_i gen$'s complexity. Notice that the overall number N of examples required per a single iteration is bounded by

$$\tilde{O}\left(\frac{T_{upper}^2}{\varepsilon^2} + Q[WL(\gamma_{WL} = \frac{\delta}{4T_{upper}})]\right).$$

As follows from $D_i gen$'s construction, when it receives an instance x from EX,

$$\Pr\left[x \textit{ is chosen}\right] = \frac{w(x)}{\frac{27 T_{upper} Z_i}{\varepsilon}} \geq \frac{w(x)}{\frac{81 T_{upper} \overline{Z_i}}{2\varepsilon}} .$$

Therefore, the probability P_{D_i} that $D_i gen$ returns a result in any single iteration is

$$P_{D_i} = \sum_{x \in X} D(x) \cdot \Pr\left[x \textit{ is chosen}\right] \geq \frac{2\varepsilon}{81 T_{upper} \overline{Z_i}} \cdot \sum_{x \in X} (w(x) \cdot D(x)) = \frac{2\varepsilon}{81 T_{upper}} .$$

The expected total number of "trials" to receive N examples is at most

$$\tilde{O}\left(\frac{1}{\varepsilon \gamma_{\min}^2} \cdot \left(\frac{1}{\gamma_{\min}^4 \varepsilon^2} + Q[WL]\right)\right),$$

still assuming logarithmic dependence of $WL(\gamma_{WL})$'s complexity on γ_{WL}. Multiplying this expression by (12) gives the desired query complexity bound.

Each $D_i gen$'s "trial" requires some $w(x)$'s value, therefore the overall time consumed by $D_i gen$ during *QuickFilt*'s execution equals

$$\tilde{O}\left(\frac{T^2}{\varepsilon \gamma_{\min}^2} \cdot \left(Q[WL] + \frac{1}{\gamma_{\min}^4 \varepsilon^2}\right)\right).$$

The summarizing expression for the time complexity of *QuickFilt* is

$$\tilde{O}\left(\frac{T^2}{\varepsilon \gamma_{\min}^2} \cdot Q[WL] + \frac{T^2}{\varepsilon^3 \gamma_{\min}^6} + T \cdot T[WL]\right).$$

Using Equation (12), we get the required time complexity bound, which completes the proof.
■ *Theorem 5*

Notice that *QuickFilt*'s time complexity depends on two different "γ-related" attributes: $\gamma_{\min}^2$ and $\mu_{1\le i\le T}\left[\gamma_i^2\right]$. In fact, in both cases $\gamma_{\min}^2$ is used only to pre-estimate the overall number of boosting iterations and may be replaced by any lower bound for $\mu_{1\le i\le T}\left[\gamma_i^2\right]$.

Also notice that the query complexity may be significantly reduced by reuse of the same set of instances throughout the iterations. That can be done because an *oblivious* sampling is used by the algorithm: the samples are chosen without regard to the target function. Notice that in this case the boosting analysis is still applied to the whole set of instances, and aside from the additional memory required to store the set of examples, this approach entirely corresponds to boosting in filtering mode.

4.2 Original *AdaBoost* is not smooth.

We give a simple proof for the fact that the distributions provided by *AdaBoost* are not smooth if the algorithm is executed on a domain of exponential size.

Claim 6 *Suppose that AdaBoost was executed on uniform distribution D over domain X of size 2^n and provided with accuracy parameter ε. Suppose also that each time WL was called it provided at least γ-accurate weak hypothesis. Then there exists a responses scheme for WL so that one of the distributions provided by AdaBoost would not be smooth. Namely, the weight of a specific point $x_0 \in X$ w.r.t. the same distribution would be either* $\mathbf{const}^{1/\gamma} \cdot D(x_0)$ *(for a fixed* $\mathbf{const} > 1$*) or at least $\gamma \cdot 2^n \cdot D(x_0)$.*

Proof of Claim 6 Suppose that all the weak hypotheses received by *AdaBoost* are of accuracy exactly γ. In this case the booster assigns the same weights to all the weak hypotheses in the generated final hypotheses; moreover, the weight update factor β used by *AdaBoost* constantly equals $\frac{1/2-\gamma}{1/2+\gamma}$.

We may apply Freund's result (1992), showing that there exists a behavioral scheme for the weak learner which satisfies the requirement $\gamma_i \equiv \gamma$ and forces the booster to make

$$\frac{1}{2}\gamma^{-2}\log(\varepsilon^{-1})$$

iterations. As well, all the produced weak hypotheses are (± 1)-valued. For the rest of the proof, we assume that $\gamma_i \equiv \gamma$. Note that we should not care about the representation length of the weak hypotheses (since no assumptions are made regarding the "generality" of the weak learner in the standard analysis of *AdaBoost*).

Let WL' be a weak learner following the mentioned scheme for the given ε, for $\gamma' \triangleq 2\gamma$ and for the domain $X' \triangleq X \setminus \{x_0\}$. We will use another weak learner WL: it always creates a hypothesis h following WL' on X' and giving the incorrect (binary) value for x_0. Further, we tune the resulting overall accuracy: If it is greater than γ then we negate some correct labels of points in X', thus bringing the accuracy to γ, if that is possible. If that cannot be performed, this means that the individual weight of x_0 is more than γ and $D_i(x_0) \ge \gamma \cdot 2^n \cdot D(x_0)$.

The booster needs to bring the accuracy of the majority vote over X' to accuracy at least $1-\varepsilon$, for that it must perform at least

$$T \triangleq \frac{1}{2}\gamma'^{-2}\log(\varepsilon^{-1}) = \frac{1}{8}\gamma^{-2}\log(\varepsilon^{-1})$$

iterations, as follows from Freund's lower bound. As follows from the analysis of *AdaBoost*, for the last iteration the "mean" of the weights of all the points is at most $(1-2\gamma)^{T-1}$, while the individual weight of x_0 remains unchanged ($\equiv 1$), and therefore $D_T(x_0) \geq \mathbf{const}^{\gamma^{-1}}$ for some $\mathbf{const} > 1$, Q.E.D.

$\blacksquare_{\textit{Claim 6}}$

5. Adaptiveness versus Smoothness, or Why Do We Call *QuickFilt* "*AdaBoost*"?

The smoothing technique we introduced does not "dramatically" affect the number of iterations performed by the algorithm: If we recall Proposition 1, we may note that in order for the "new" final error to be bounded by ε, it suffices to bound both the "old" final error and $\frac{T}{p}$ by $\varepsilon/2$. This is done (up to certain constants) in Section 4 (using T_{upper} which is an upper bound on the number of boosting iterations). The smoothness parameter which results from this approach is $O(T_{upper}\varepsilon^{-1})$ and the number of performed iterations remains unchanged (when evaluated in terms of O). Since the number of iterations usually defines uniquely the size of the final hypothesis, the latter characteristic remains unaffected as well. For instance, this situation occurs in the case of *AdaBoost*.

On the other hand, the influence of our modifications becomes apparent in terms of the complexity of a single iteration, which is increased.

Let us now try to advocate the claim that we are actually "extending" the range of applicability of the *original* algorithm *AdaBoost*. One of the distinctive valuable features of *AdaBoost* is its *adaptiveness*. Informally, we say that a boosting algorithm is adaptive if it utilizes the "additional advantage" of those weak hypotheses whose actual accuracy is higher than the corresponding lower bound (denoted above by $\gamma_{\min}$). This general intuition may be formalized as follows.

First of all, let us clarify what we mean by saying that some quantity is adaptive in the parameters γ_i of all the weak hypotheses h_i which were received from WL during the boosting process. For our current needs, it suffices to say that a quantity is adaptive when it depends polynomially upon the value of $\mu_{1\leq i\leq T}\left[\gamma_i^2\right]$; on the other hand, by saying that a quantity is non-adaptive we mean that it depends on the value of $\gamma_{\min}$ (if this dependence is either polynomial or logarithmic we accordingly say that the quantity is polynomially or logarithmically non-adaptive).

We give two possible definitions for adaptiveness, which differ between them in their "strictness". The more flexible requirement – we denote it by (I) – is that the number of iterations performed by the booster is adaptive in γ_i's. The second (more strict) requirement (denoted by (II)) is that algorithm, being used to learn a concept over an exponentially large domain, runs in overall adaptive time.

In the case of *AdaBoost* (both in its original form and under our modification), the number of iterations, being expressed in terms of $\mu_{1\leq i\leq T}\left[\gamma_i^2\right]$ and ε, does not depend on

$\gamma_{\min}$. However, for the original *AdaBoost* the parameter ε is not an "independent" one, its value is inversely linear in the size of the learning sample. The latter should obviously chosen a priori; moreover, since we are willing to bound the generalization error, the required size of the learning sample depends upon the size of the final hypothesis and therefore upon $\mu_{1\leq i\leq T}\left[\gamma_i^2\right]$ as well. Therefore we have to find a lower bound for $\mu_{1\leq i\leq T}\left[\gamma_i^2\right]$ before the learning is even started, which obviously cannot be done adaptively and has to rely upon the "worst case assumption" for the values of γ_i's, namely upon $\gamma_{\min}$. Since the booster's behavior is not "random", it is not clear how a "statistical" lower bound for $\mu_{1\leq i\leq T}\left[\gamma_i^2\right]$ which would be significantly better than $\gamma_{\min}^2$ may be found.

In the case of *AdaBoost*, however, the non-adaptiveness in sense (I) is only logarithmic (since logarithmic is dependence of *AdaBoost*'s complexity on ε).

Since the above considerations in fact apply to any boosting algorithm working by sampling whose final hypothesis size depends on γ_i's (in either straightforward or adaptive way), we claim that any such algorithm cannot be adaptive (when executed over a domain of super-polynomial size), even w.r.t. the definition (I) (and obviously not w.r.t. (II), which is "strictly stronger" than (I)).

On the other hand, as follows from Theorem 5, *QuickFilt* is adaptive w.r.t. (I); however we note that *QuickFilt* does not satisfy the requirement (II)(Gavinsky, 2002, describes an algorithm satisfying (II)).

As mentioned before, *QuickFilt* depends on $\gamma_{\min}$ because it needs an a priori bound on the number of iterations to be performed (exactly like *AdaBoost*, when facing the need to determine the size of the training set). Consequently, *QuickFilt*'s non-adaptiveness is of the same "origin" as that of *AdaBoost*; on the other hand, unlike *AdaBoost*, *QuickFilt* is fully adaptive w.r.t. the definition (I) above.

6. Learning DNF in the PAC-Model with Membership Oracle under Polynomially Near-Uniform Distribution

Jackson (1997) had posed the question of whether *AdaBoost* may be applied to solve the task of uniform DNF learning with membership queries.

First of all, let us introduce the corresponding learning model – *PAC with membership queries*. It may be viewed as an "extended" PAC model (considered before). The extension is as follows: a learner is allowed to query the value of the target function f in a specific point x (or, in other words, to query the membership of x in the set $\{x \in X | f(x) = 1\}$).

Second, let us try to understand better the meaning of Jackson's question. Roughly speaking, there are two "problems" with *AdaBoost*: it cannot work in the filtering mode and it is not smooth. On the other hand, there are some potential benefits in using an adaptive booster (like *AdaBoost*): While in DNF learning a polynomially small lower bound may be provided for the values of γ_i's, it seems that in many cases this bound can be noticeably surpassed in practice during specific iterations of boosting. Therefore, adaptiveness, though not essential for efficient learning, is obviously a pleasurable advantage.

It would be useful to figure out what kind of smoothness should we anticipate from the possible application of *AdaBoost*? Using the smoothness gradation introduced in Section 5, let us recall that (II) was shown to fail for *AdaBoost* – unfortunately we have to limit ourselves to (I) only (Gavinsky, 2002, describes a boosting algorithm satisfying (II)). The application of our technique makes it possible to execute *AdaBoost* in filtering mode: Below we construct a learner which "properly" satisfies (I) and produces a final hypothesis of adaptive size (unlike original *AdaBoost* which is logarithmically non-adaptive even w.r.t. (I)).

While we've chosen the problem of DNF learning for the purpose of demonstrating the smoothing technique, there are (as mentioned before) some other problems which are solved using (essentially) smooth boosting, like noise-tolerant learning (Freund, 1999, Domingo and Watanabe, 2000, Servedio, 2001), learning via extended statistical queries (Bshouty and Feldman, 2001), agnostic learning (Ben-David, Long and Mansour, 2001, Gavinsky, 2002) and maybe some others. Using our technique for applying smooth *AdaBoost* the (potentially pleasurable) feature of adaptiveness may be added to all those solutions.

But let us return to the problem of DNF learning. We assume that $X = \{0,1\}^n$ and that the target distribution D of the booster is polynomially near-uniform, satisfying

$$\forall x \in X: \; D(x) \leq \frac{1}{2^n} \cdot \alpha_D,$$

for some α_D. As follows from *QuickFilt*'s analysis (Theorem 5), in this case the distributions the booster produces are $\tilde{O}(\varepsilon^{-1}\gamma_{\min}^{-2} \cdot \alpha_D)$-near uniform.

In Jackson's paper (Jackson, 1997), the algorithm used for weak DNF membership learning was based on algorithm KM, introduced by Kushilevitz and Mansour (1993). In this paper we use a more recent algorithm introduced by Bshouty, Jackson and Tamon (1999) (denoted by W), which was developed from another algorithm described by Levin (1993). The algorithm W has, in the context of DNF-learning, certain complexity advantages over the originally used KM (in particular, W was used by Klivans and Servedio (1999) to construct the most efficient algorithm for DNF-learning known so far).

Like its predecessors, W is capable of finding "heavy" Fourier coefficients of the target w.r.t. the uniform distribution. In this case, the basis for Fourier transform[4] coincides with the set of all the parity functions:

$$\forall A \subseteq \{1, ..., n\}: \; \chi_A \triangleq (-1)^{\sum_{i \in A} x_i} .$$

The following statement is taken from Jackson (1997):

Fact 7 *For every DNF expression f with s terms and for every distribution D on the instance space of f, there exist a parity function χ_A so that* $\mu_{x \sim D}\left[\frac{|f(x) - \chi_A(x)|}{2}\right] \leq \frac{1}{2} - \frac{1}{4s+2}$.

Although the aforementioned algorithms for weak DNF learning using parity functions are capable of learning only under the uniform distribution, a simple trick (Jackson, 1997) may be used to overcome this difficulty. The resulting performance of the weak learners depends critically on the smoothness of the target distribution; moreover it seems that weakly learning DNF (even given an access to the membership oracle) over arbitrary target distributions is hard (see Jackson, 1997, for a discussion). So, in this case the smoothness limitation being put on a boosting algorithm results from the qualities of the weak learner.

4. For an overview of Fourier analysis applications to Machine Learning, see, for example, Mansour (1994).

In order to cope with the ("strict") uniformity limitation, the following reduction is used. Instead of weakly learning the target binary function $f(x)$ under some non-uniform distribution D_i, we will ask WL to weakly approximate w.r.t. the uniform ($\mathbf{U}$) the following (real-valued) function $g(x)$:

$$g(x) \triangleq 2^n D_i(x) \cdot f(x).$$

It is shown by Jackson (1997) that this function has "heavy" Fourier coefficients which weakly approximate $f(x)$ over the distribution D_i. Moreover, the quantitative value of $\mu_{\mathbf{U}}\left[\chi'(x)g(x)\right]$ (which is the coefficient corresponding to χ' in $g(x)$'s Fourier expansion) satisfies

$$\mu_{\mathbf{U}}\left[\chi'(x)g(x)\right] = \mu_{x\sim D_i}\left[\chi'(x)f(x)\right] = 1 - \mu_{x\sim D_i}\left[|f(x) - \chi'(x)|\right].$$

We conclude that in the case of weak DNF approximation using parity functions, the value of $\gamma_{\min}$ is bounded below by $\Omega(s^{-1})$ (recall that s is the number of DNF-terms contained in the formula). Algorithm W requires a lower bounds θ for the magnitude of Fourier coefficients to be found, which in our case corresponds to twice the lower bound on $\gamma_{\min}$, as stated above.

We provide W with a membership oracle for the function

$$g'(x) \triangleq \frac{g(x)}{\max_x\{2^n D_i(x)\}}$$

(which can be easily simulated knowing $f(x)$, $D_i(x)$ and $\max_x\{2^n D_i(x)\}$) and with

$$\theta' \triangleq \frac{\theta}{\max_x\{2^n D_i(x)\}}.$$

Algorithm W requires a "distribution oracle": during the i'th iteration, the booster should be able to report what probabilistic weight it assigned by D_i to a specific point $x \in \{0,1\}^n$, up to some constant multiplicative factor. As mentioned in Subsection 4.1.1, such estimation is possible for $QuickFilt$. Jackson shows that an estimation is sufficient and it doesn't adversely affect the performance of KM; the same holds for W that we use.

Denote $d_\infty(D_i) \triangleq \max_x\{2^n D_i(x)\}$. As follows from the construction of $g'(x)$, it holds that

$$\theta'^{-1} = O(d_\infty(D_i) \cdot \gamma_{\min}^{-1}) = \tilde{O}\left(\varepsilon^{-1}\gamma_{\min}^{-3} \cdot \alpha_D\right) = \tilde{O}\left(\frac{s^3\alpha_D}{\varepsilon}\right),$$

using the bound on $\gamma_{\min}$ provided by Fact 7. The algorithm W's complexity depends on θ' and equals $\tilde{O}(n\theta'^{-2})$, which leads us to

$$Q[W] = T[W] = \tilde{O}\left(\frac{ns^6\alpha_D^2}{\varepsilon^2}\right).$$

A straightforward "filtering" application of $QuickFilt$ gives us the complexity of

$$\tilde{O}\left(ns^{10}\varepsilon^{-3} \cdot \alpha_D^2(n)\right)$$

queries and

$$\tilde{O}\left(ns^{12}\varepsilon^{-3}\cdot\alpha_D^2(n)\right)$$

time.[5] As mentioned above, the query complexity may be lowered by example reuse throughout the iterations. Another possibility for query complexity improvement comes from the fact that now we are given a membership oracle. Therefore, no query is in fact needed when a point is "rejected" by the booster's filtering scheme and the query complexity of receiving an example from $D_i gen$ may be reduced to a single (membership) query.

As a result of our application, the boosting algorithm is (I)-adaptive, it performs

$$O\left(\frac{\ln(\varepsilon^{-1})}{\mu_{1\le i\le T}\left[\gamma_i^2\right]}\right)$$

iterations and produces a final hypothesis of the same size.

Acknowledgments

We would like to thank COLT 2001 conference participants whose comments contributed to the evolution of this essay.

References

N. Bshouty and V. Feldman. On Using Extended Statistical Queries to Avoid Membership Queries. *Proceedings of the 14th Annual Conference on Computational Learning Theory, pp. 529-545*, 2001.

N. Bshouty, J. Jackson and C. Tamon. More efficient PAC-learning of DNF with membership queries under the uniform distribution. *Proceedings of the 12th Annual Conference on Computational Learning Theory, pp. 286-295*, 1999.

S. Ben-David, P. M. Long and Y. Mansour. Agnostic Boosting. *Proceedings of the 14th Annual Conference on Computational Learning Theory, pp. 507-516*, 2001.

C. Domingo and O. Watanabe. MadaBoost: A modification of AdaBoost. *Proceedings of the 13th Annual Conference on Computational Learning Theory, pp. 180-189*, 2000.

Y. Freund. Boosting a weak learning algorithm by majority. *Proceedings of the 3th Annual Conference on Computational Learning Theory, pp. 202-216*, 1990.

Y. Freund. An improved boosting algorithm and its implications on learning complexity. *Proceedings of the 5th Annual Conference on Computational Learning Theory, pp. 391-398*, 1992.

Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation 121(2), pp. 256-285*, 1995.

5. The most efficient non-adaptive result known so far has been achieved by Klivans and Servedio (1999).

Y. Freund. An adaptive version of the boost by majority algorithm. *Proceedings of the 12th Annual Conference on Computational Learning Theory, pp. 102-113*, 1999.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences 55(1), pp. 119-139*, 1997.

D. Gavinsky. Optimally-Smooth Adaptive Boosting and Application to Agnostic Learning. *Proceedings of the 13th International Conference on Algorithmic Learning Theory,* , 2002.

J. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *Journal of Computer and System Sciences 55(3), pp. 414-440*, 1997.

E. Kushilevitz and Y. Mansour. Learning Decision Trees using the Fourier Spectrum. *SIAM Journal on Computing 22(6), pp. 1331-1348*, 1993.

A. R. Klivans and R. A. Servedio. Boosting and Hard-Core Sets. *Proceedings of the 40th Annual Symposium on Foundations of Computer Science, pp. 624-633*, 1999.

M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM 41(1), pp. 67-95*, 1994.

L. Levin. Randomness and Non-determinism. *Journal of Symbolic Logic 58(3), pp. 1102-1103*, 1993.

Y. Mansour. Learning Boolean Functions via the Fourier Transform. *Theoretical Advances in Neural Computing and Learning, Kluwe Academic Publishers,* , 1994.

R. E. Schapire. The strength of weak learnability. *Machine Learning 5(2), pp. 197-227*, 1990.

R. Servedio. Smooth Boosting and Learning with Malicious Noise. *Proceedings of the 14th Annual Conference on Computational Learning Theory, pp. 473-489*, 2001.

L. Valiant. A theory of learnable. *Communications of the ACM 27(11), pp. 1134-1142*, 1984.