

Lecture 9-10: Learning DNF in  $PAC_U(MQ)$ 

Authors: Prof. Nader Bshouty

Scribe: Rotem Bennet

## 1 Introduction

The question of whether DNF formulas can be learned in the PAC model, is one of the most studied questions in the field of computational learning. Many negative results have been published since this question was first posed by Valiant in 1984, and until recently it was not known whether DNF can even be learned with the help of membership queries. This was the situation until Jackson made the breakthrough, by showing an algorithm for PAC learning DNF with membership queries over the uniform distribution. His result was achieved using a combination of Kushilevitz and Mansour's algorithm (already discussed in previous lectures and is denoted here by [KM]), and Freund's boosting method (that is presented later, in section 3). In the current lecture we shall walk along Jackson's path, with the exception of some minor changes to his original article which were made for the sake of simplicity.

For the rest of this lecture, let  $X = \{0, 1\}^n$ . Let  $D$  be some given distribution,  $C$  be a class of functions and  $f \in C$  be the target function. When  $f$  is a DNF formula, we denote its size (number of terms) by  $s$ .

**Definition 1** *Polynomially "near-uniform" distribution  $D$  is a distribution in which for all instances  $x \in \{0, 1\}^n$*

$$D(x) \leq \frac{\text{poly}\left(s, \frac{1}{\epsilon}\right)}{2^n}$$

## 2 Weakly learning DNF over near-uniform distributions

In this section we shall prove the weak learnability of DNF over near-uniform distributions, which would be boosted later to produce the desired strong learnability over the uniform distribution.

**Theorem 2 (Jackson)** *DNF is weakly-learnable in  $PAC_D(MQ)$  over any near-uniform distribution  $D$ .*

**Proof:** As proven in the previous lecture, any DNF with  $s$  terms has an  $a \in \{0, 1\}^n$  such that:

$$|E_D [f\chi_a]| \geq \frac{1}{4s}$$

When writing the expectation explicitly, we get:

$$|E_D [f\chi_a]| = \left| \sum_x D(x)f(x)\chi_a(x) \right| = \left| \frac{1}{2^n} \sum_x 2^n D(x)f(x)\chi_a(x) \right| = |E_U [2^n D(x)f(x)\chi_a(x)]| \geq \frac{1}{4s}$$

Denoting  $2^n D(x)f(x)$  by  $g(x)$ , we get that the last equation can also be written as

$$|\hat{g}_a| \geq \frac{1}{4s}$$

The algorithm of [KM], for finding large fourier-coefficients, runs in time  $\text{poly}(|g|, s, n, \frac{1}{\epsilon}, \frac{1}{\delta})$ , and thus, if  $|g|$  is polynomially bounded, then we can use [KM] to find the  $\chi_a$  for which  $|\hat{g}_a| \geq \frac{1}{4s}$ , and this  $\chi_a$  is the desired weak-approximation of  $f$ . Now

$$|g(x)| = |2^n D(x)f(x)| \leq 2^n \frac{\text{poly}(s, \frac{1}{\epsilon})}{2^n} = \text{poly}\left(s, \frac{1}{\epsilon}\right)$$

as required. ■

We have now proved that DNF can be learned weakly over distributions that are near-uniform. Our final aim is proving that DNF is PAC (*strong*) learnable over *Uniform*, meaning we should strengthen the learnability results but we may limit it to Uniform only. This would be done in two steps: first we prove that weak-learnability over *any* distribution implies strong-learnability, and then we use a variation of it to learn DNF over the uniform distribution, taking advantage of the above weak learnability result over near-uniform distributions.

### 3 Weak implies Strong (!)

**Theorem 3** *Let  $C$  be a class of boolean functions. If  $C$  is Weak PAC-learnable over any distribution then  $C$  is PAC-learnable over any distribution.*

**Proof:** Let **WL** be the weak-learning algorithm of the function class  $C$ . We shall construct an algorithm **AdaBoost** that “boosts” the weak-learner **WL** to produce a strong learning as required. It learns the target function several times using **WL**, each time on a different distribution, thus producing several different weak hypotheses. The distributions are tailored dynamically between the learning sessions of the algorithm, with the purpose of adaptively correcting past mistakes of the previous hypotheses. After “enough” iterations, **AdaBoost** outputs a hypothesis that is a majority vote over all the learned hypotheses.

**Algorithm AdaBoost** Let  $h_0$  be the hypothesis that **WL** outputs after weak-learning  $f$  over  $D$ . From the definition of weak learning, we have

$$\Pr_D [f = h_0] \geq \frac{1}{2} + \gamma$$

where  $\gamma \geq \frac{1}{\text{poly}(n,s)}$ . We initialize a set of weak hypotheses:

$$H_1 \leftarrow \{h_0\}$$

We define a constant  $\alpha$  to be

$$\alpha \stackrel{\text{def}}{=} \frac{1 - 2\gamma}{1 + 2\gamma} = 1 - \frac{4\gamma}{1 + 2\gamma} < 1$$

This constant determines (as we shall see later) the extent of changing the distribution in each step, which consequently affects the speed and accuracy of the whole algorithm. The number of algorithm’s iterations  $k$  is set to be:

$$k \stackrel{\text{def}}{=} \frac{1}{2\gamma^2} \ln \frac{1}{\epsilon}$$

In each of the  $k$  iterations we do the following:

- $\forall x \in X$ , we denote the number of correct hypotheses in  $H_r$  by  $n_r(x)$ :

$$n_r(x) \stackrel{\text{def}}{=} |\{h \in H_r : h(x) = f(x)\}|$$

- We define a new distribution  $D_r$  which puts more weight on instances which are misclassified by many hypotheses (instances with low  $n_r(x)$ ). This is done by defining for all  $x \in X$ :

$$D_r(x) \stackrel{\text{def}}{=} \frac{\alpha^{n_r(x)} \cdot D(x)}{L_r}$$

where  $L_r$  is the normalization factor (which makes  $D_r$  a distribution, i.e. summed to 1):

$$L_r \stackrel{\text{def}}{=} \sum_x \alpha^{n_r(x)} \cdot D(x)$$

- We use **WL** to weak-learn  $f$  with a simulated oracle  $\text{EX}(f, D_r)$ . The output hypothesis  $h_r$  is added to  $H_r$ , the set of previous weak hypotheses: The new set is denoted by  $H_{r+1}$ .

(The practical aspects of simulating the distribution  $D_r$  are discussed later in this section).

After the  $k^{\text{th}}$  iteration, the algorithm outputs  $\text{Maj}(H_k)$ , which is a majority vote over the hypotheses in  $H_k$ , formally defined as:

$$\forall x \in X, \text{Maj}(H_k)(x) \stackrel{\text{def}}{=} \begin{cases} +1 & |\{h \in H_k : h(x) = +1\}| > \lfloor \frac{|H_k|}{2} \rfloor \\ -1 & \text{otherwise} \end{cases}$$

Now we are left with the task of proving that the output hypothesis  $\text{Maj}(H_k)$  is indeed an  $\epsilon$ -approximation of the target function  $f$ , which means **AdaBoost** PAC-learns  $f$  in the *strong* sense.

**Lemma 4**  $\Pr_D [f(x) \neq \text{Maj}(H_k)(x)] \leq \epsilon$

**Proof:** We shall first make some observations about the  $r^{\text{th}}$  iteration, which would help us get the feeling of what is needed in-order to complete the proof.

**Definition 5** We denote by  $X_i^{(r)}$  ( $0 \leq i \leq r$ ) the set:

$$X_i^{(r)} = \{x_0 \in X | n_r(x_0) = i\}$$

(Note that for a given instance  $(x_0, f(x_0))$ , the learner can easily compute  $n_r(x_0)$ ).

Continuing the above definition we can make the following observations:

- The sets  $X_0^{(r)}, \dots, X_r^{(r)}$  are a *partitioning* of  $X$ , i.e. each  $x \in X$  is in exactly one set (according to the number of hypotheses that correctly classify it).
- For any instance  $x$  in the sets  $X_0^{(r)}, \dots, X_{\lfloor \frac{r}{2} \rfloor}^{(r)}$ ,  $\text{Maj}(H_r)$  is mistaken! ( $\text{Maj}(H_r)(x) \neq f(x)$ ). For the rest of the instances  $\text{Maj}(H_r)$  is correct.
- The normalization factor  $L_r$  can now be rewritten as

$$L_r = \sum_{i=0}^r \alpha^i D(X_i^{(r)})$$

(We can sum over all the instances  $x \in X_i^{(r)}$  together, since they all have the same  $n_r(x)$  ( $= i$ ), and therefore their weights are multiplied by the same factor  $\alpha^i$ ).

- From the previous observations we may conclude that the error after the  $r^{th}$  iteration is:

$$\epsilon_r \stackrel{\text{def}}{=} Pr_D [f \neq Maj(H_r)] = \sum_{i=0}^{\lfloor \frac{r}{2} \rfloor} D(X_i^{(r)})$$

- When comparing the partitioning of  $X$  according to  $n_r$  and  $n_{r+1}$ , we observe that the instances in  $X_i^{(r)}$  are divided to those that are correctly classified by  $h_r$ , which become part of  $X_{i+1}^{(r+1)}$  (we will denote these instances by  $S_i^{(r)}$ ), and those that are wrongly classified by  $h_r$  and become part of  $X_i^{(r+1)}$ .

This can be expressed as:

$$X_i^{(r+1)} = (X_i^{(r)} \setminus S_i^{(r)}) \cup S_{i-1}^{(r)}$$

In Figure 3.1 we can view graphically the partitioning of the instances in  $X$  along the algorithm's iterations.

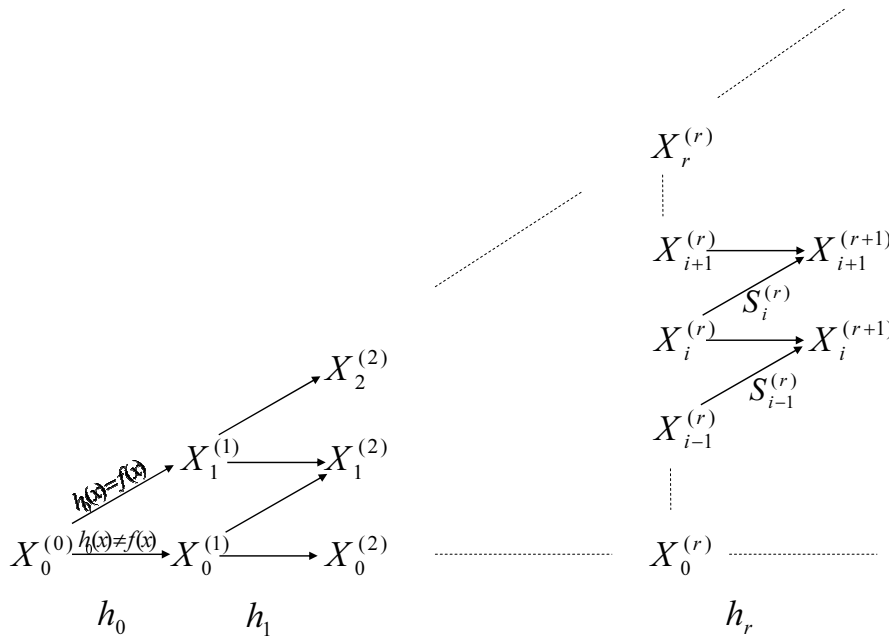


Figure 3.1: Partitioning of the instances  $x \in X$  according to  $n_r(x)$ .

- From the definition of  $S_i^{(r)}$  we get that

$$Pr_{D_r} [h_r = f] = \sum_{i=0}^r D_r(S_i^{(r)})$$

Since  $h_r$  is a weak hypothesis of  $f$ , then the above probability is  $\geq \frac{1}{2} + \gamma$ . When writing the sum in

terms of the original distribution ( $D$ ), we get that:

$$\sum_{i=0}^r D_r \left( S_i^{(r)} \right) = \sum_{i=0}^r \alpha^i \frac{D \left( S_i^{(r)} \right)}{L_r} \geq \frac{1}{2} + \gamma \quad (3.1)$$

The following claim will prove that the algorithm's error decreases exponentially as a function of the number of iterations, and thus, by choosing the right polynomial number of iterations  $k$ , we limit the error of the output hypothesis to  $\epsilon$ , as required.

**Claim 6** *At the  $r^{\text{th}}$  iteration of the AdaBoost algorithm:*

1.  $\alpha^{\lfloor \frac{r}{2} \rfloor} \cdot \epsilon_r \leq L_r \leq (1 - 2\gamma)^r$
2.  $\epsilon_r \leq (1 - 4\gamma^2)^{\frac{r}{2}}$
3. *After the  $k^{\text{th}}$  iteration:  $\epsilon_k \leq \epsilon$*

**Proof:** We shall first show ( $1 \Rightarrow 2$ ), which implies (3) by substituting  $k$ , and then we complete the proof by proving (1).

( $1 \Rightarrow 2$ ): From (1) we have

$$\epsilon_r \leq \frac{(1 - 2\gamma)^r}{\alpha^{\lfloor \frac{r}{2} \rfloor}} \leq \frac{(1 - 2\gamma)^r}{\alpha^{\frac{r}{2}}}$$

substituting  $\alpha = \frac{1-2\gamma}{1+2\gamma}$

$$\epsilon_r \leq \frac{(1 - 2\gamma)^r (1 + 2\gamma)^{\frac{r}{2}}}{(1 - 2\gamma)^{\frac{r}{2}}} = (1 - 4\gamma^2)^{\frac{r}{2}}$$

as required. Since  $(1 + \frac{x}{n})^n \leq e^x$  we have

$$\epsilon_r \leq e^{\frac{-r}{2\gamma^2}}$$

and for:  $r = k = \frac{1}{2\gamma^2} \ln \frac{1}{\epsilon}$ , we get

$$\epsilon_k \leq \epsilon$$

which proves (3).

(1): In-order to ease our use of summations, we shall define:

$$X_{r+1}^{(r)} = S_{r+1}^{(r)} = S_{-1}^{(r)} = \emptyset$$

We begin with the right-hand side of the equation. We would like to get a relation between the size of the normalization-factor at one iteration ( $L_r$ ) and its size in the next iteration ( $L_{r+1}$ ) at the form of a recursive formula. The basis of such a recursion is:  $L_0 = 1$ , since  $D_0 = D$  and  $D$  is already a distribution, i.e. it is already normalized. After having the desired formula, we can calculate an upper bound for  $L_r$  in non-recursive terms.

Writing  $L_{r+1}$  according to its definition

$$L_{r+1} = \sum_{i=0}^{r+1} \alpha^i D \left( X_i^{(r+1)} \right)$$

we have observed earlier that  $X_i^{(r+1)}$  can be written as a function of the previous iteration

$$L_{r+1} = \sum_{i=0}^{r+1} \alpha^i \left( D \left( X_i^{(r)} \right) - D \left( S_i^{(r)} \right) + D \left( S_{i-1}^{(r)} \right) \right)$$

by definition:  $\sum_{i=0}^{r+1} \alpha^i D \left( X_i^{(r)} \right) = L_r$ , so we can separate it from the summation and get

$$L_{r+1} = L_r + \sum_{i=0}^{r+1} \alpha^i \left( -D \left( S_i^{(r)} \right) + D \left( S_{i-1}^{(r)} \right) \right)$$

From looking at the above sum, one gets the impression that consecutive parts of it cancel each other in some way. Since the summed parts are multiplied by  $\alpha^i$ , they do not cancel completely, but are only subtracted from each other. Let us look at a small part of this chain of summings (for  $i = j - 1, j, j + 1$ ):

$$\begin{aligned} \dots + \alpha^{j-1} \left( -D \left( S_{j-1}^{(r)} \right) + D \left( S_{j-2}^{(r)} \right) \right) + \alpha^j \left( -D \left( S_j^{(r)} \right) + D \left( S_{j-1}^{(r)} \right) \right) + \alpha^{j+1} \left( -D \left( S_{j+1}^{(r)} \right) + D \left( S_j^{(r)} \right) \right) \dots = \\ \dots + \alpha^{j-1} D \left( S_{j-2}^{(r)} \right) + (\alpha^j - \alpha^{j-1}) D \left( S_{j-1}^{(r)} \right) + (\alpha^{j+1} - \alpha^j) D \left( S_j^{(r)} \right) - \alpha^{j+1} D \left( S_{j+1}^{(r)} \right) \dots = \\ \dots + \alpha^{j-1} D \left( S_{j-2}^{(r)} \right) + (\alpha - 1) \alpha^{j-1} D \left( S_{j-1}^{(r)} \right) + (\alpha - 1) \alpha^j D \left( S_j^{(r)} \right) - \alpha^{j+1} D \left( S_{j+1}^{(r)} \right) \dots \end{aligned}$$

From the above we can see that we remain with

$$L_{r+1} = L_r + (\alpha - 1) \sum_{i=0}^r \alpha^i D \left( S_i^{(r)} \right)$$

By (3.1)

$$\sum_{i=0}^r \alpha^i \frac{D \left( S_i^{(r)} \right)}{L_r} \geq \frac{1}{2} + \gamma$$

so now we can substitute it, and substitute  $\alpha$  in our equation, and get the desired result:

$$L_{r+1} \leq L_r + (\alpha - 1) \left( \frac{1}{2} + \gamma \right) L_r = L_r - \frac{4\gamma}{1 + 2\gamma} (1 + 2\gamma) \frac{1}{2} L_r = (1 - 2\gamma) L_r$$

and now, since  $L_0 = 1$ , then from the above recursion we get

$$L_r \leq (1 - 2\gamma)^r$$

As required.

Now we shall prove the left-hand side of the equation, which is quite immediate. By definition

$$L_r = \sum_{i=0}^r \alpha^i D \left( X_i^{(r)} \right) \geq \sum_{i=0}^{\lfloor \frac{r}{2} \rfloor} \alpha^i D \left( X_i^{(r)} \right) \geq \alpha^{\lfloor \frac{r}{2} \rfloor} \sum_{i=0}^{\lfloor \frac{r}{2} \rfloor} D \left( X_i^{(r)} \right) = \alpha^{\lfloor \frac{r}{2} \rfloor} \epsilon_r$$

which completes the proof of Claim 6. ■

Since  $\epsilon_k \stackrel{\text{def}}{=} Pr_D [f \neq Maj(H_k)] = \sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} D \left( X_i^{(k)} \right)$ , then from Claim 3 we get that  $Maj(H_k)$  is an  $\epsilon$ -approximation of  $f$  as required. This completes the proof of Lemma 4. ■

We may now look at the **AdaBoost**( $\delta, \epsilon, D$ ) algorithm:

- 1:  $k \leftarrow \frac{1}{2\gamma^2} \ln \frac{1}{\epsilon}$
- 2:  $h_0 \leftarrow \mathbf{WL}(\delta, EX(f, D))$
- 3:  $H_1 \leftarrow \{h_0\}$
- 4: **For**  $r = 1, \dots, (k - 1)$  **do**
- 5:    $\forall x \in X, n_r(x) \stackrel{\text{def}}{=} |\{h \in H_r : h(x) = f(x)\}|$
- 6:    $L_r \stackrel{\text{def}}{=} \sum_x \alpha^{n_r(x)} \cdot D(x)$
- 7:    $\forall x \in X, D_r(x) \stackrel{\text{def}}{=} \frac{\alpha^{n_r(x)} \cdot D(x)}{L_r}$
- 8:    $h_r \leftarrow \mathbf{WL}(\delta, EX(f, D_r))$
- 9:    $H_{r+1} \leftarrow H_r \cup \{h_r\}$
- 10: **end For**
- 11: **Output**  $Maj(H_k)$

**Simulating the distributions.** In the **AdaBoost** algorithm, the learner should, in some way, simulate distributions  $D_r$  that are specific transformations of the original distribution  $D$ . What is still unclear is how can this be done practically? Given that the initial distribution  $D$  is not known a-priori, it seems not trivial at all to set a specific probability to each instance and simulate the oracle  $EX(f, D_r)$  exactly according to these probabilities.

Although there is a way, under certain constraints, to simulate these distributions efficiently, apparently there is no need to actually do it! The intuition behind our solution goes like this: we will decrease the “effective size” of  $X$  by learning with a distribution that gives probability 0 to most of its instances, and probability  $> 0$  only to the instances in a sample given to the algorithm by  $EX(f, D)$ . Since **AdaBoost** changes the distribution in each iteration by multiplication, then any instance with probability 0 would remain with the same probability after the multiplication as well, and thus the set of relevant instances (i.e. with probability  $> 0$ ) would remain small through the whole run. We will choose a new  $\epsilon$ , small enough to make **AdaBoost** actually find a *consistent* hypothesis over the given instances (and not only PAC-learn with some error), and thus, according to what we have already seen (Occam), this hypothesis would be a strong PAC-learning of the target function over the original distribution.

More specifically, finding the desired consistent hypothesis is done as follows: after receiving the set of instances  $M$ , we shall “cheat” **AdaBoost** by setting a new distribution  $D'$  that gives weight  $\frac{1}{m}$  to each instance in  $M$ , and weight 0 to the rest of  $X$ , and setting a new  $\epsilon' = \frac{1}{2m}$ . Since the number of “relevant” instances (i.e. with probability  $> 0$ ) over  $D'$  is polynomial, and the initial distribution is known, then **AdaBoost** can simulate the distributions  $D_r$  efficiently, and output a set of hypotheses  $H_k$  such that:  $Pr_{D'} [Maj(H_k) \neq f] \leq \frac{1}{2m}$ . Since for all  $x \in M$  we have  $D'(x) = \frac{1}{m} > \frac{1}{2m}$ , then this means that  $Maj(H_k)$  is a *consistent* hypothesis for all the instances in  $M$ , and according to Occam, this gives us an  $\epsilon$ -approximation of the target function over  $D$ , as desired.

Given that we have an algorithm that outputs a weak hypothesis  $h$ , and given that the number of iterations (i.e. number of sub-functions in the final majority function) required to strengthen this hypothesis is  $k \stackrel{\text{def}}{=} \frac{1}{2\gamma^2} \ln \frac{1}{\epsilon} = \frac{1}{2\gamma^2} \ln 2m$ , then the size of the strong hypothesis is  $k \cdot |h|$ , which is sub-linear in the sample size  $m$ , and polynomial in the target function size  $s$ . Meaning there are constants  $0 \leq \alpha$  and  $0 \leq \beta < 1$ , such that the size of the final strong hypothesis is  $O(s^\alpha \cdot m^\beta)$  (which is why such an algorithm is called an  $(\alpha - \beta)$ -Occam algorithm). PAC-learning the class  $C$  using  $H$  can be done by finding a consistent hypothesis (using the  $(\alpha - \beta)$ -Occam algorithm) for a set of instances of size:

$$m = \left( \frac{2s^\alpha}{\epsilon} \right)^{\frac{1}{1-\beta}} + \frac{2}{\epsilon} \ln \frac{1}{\delta}$$

picked at random from  $X$  according to the given distribution  $D$ . As proved by Occam, the consistent

hypothesis for this set of instances is, with probability  $\geq (1 - \delta)$ , an  $\epsilon$ -approximation of the target function  $f$ , as desired.

to conclude the proof, the strong learner's algorithm  $\mathbf{SL}(\delta, \epsilon, \alpha, \beta)$  is detailed below:

- 1:  $m \leftarrow \left(\frac{2s^\alpha}{\epsilon}\right)^{\frac{1}{1-\beta}} + \frac{2}{\epsilon} \ln \frac{1}{\delta}$
- 2:  $M \leftarrow \bigcup_m EX(f, D)$
- 3:  $\forall x \in X, D'(x) \stackrel{\text{def}}{=} \begin{cases} \frac{1}{m} & x \in M \\ 0 & x \in X \setminus M \end{cases}$
- 4:  $\epsilon' \leftarrow \frac{1}{2m}$
- 5:  $h \leftarrow \mathbf{AdaBoost}(\delta, \epsilon', D')$
- 6: Output  $h$

■

## 4 Learning DNF in the $\text{PAC}_U(\text{MQ})$ model

In Theorem 3 we have seen that weak-learnability over any distribution implies strong learnability, but is the emphasis on *any* distribution necessary? When looking again into the algorithm, we find the important thing is that we can weakly-learn over distributions  $D_r$  that might be produced by **AdaBoost** during its run, as a transformation of the initial distribution  $D$ . Meaning, if we want to prove strong-learnability over the uniform distribution, then we need not prove weak-learnability over *any* distribution, but only over those distributions that might be produced by **AdaBoost** when its initial distribution is uniform. Note that this is true only if we really simulate **AdaBoost**'s distributions, and not use the method we have shown earlier to learn a consistent hypothesis over a sample  $M$  (using OCCAM), since in the later we create a completely new distribution that might be very far from the original distribution  $D$ . Thus, in the case of learning DNF, where we (currently) only want to prove learnability over Uniform, and we already have weak-learnability results over near-uniform distributions, what is left for us to do is to show that we can limit the distributions  $D_r$ , produced by **AdaBoost**, to such that are near-uniform (and then use [KM] to weak-learn the target function at each iteration of **AdaBoost**).

When learning DNF over Uniform, we would like to use, in each iteration of **AdaBoost**, the algorithm for weak-learning of DNF over near-uniform distributions. As we have seen, after  $r$  iterations, each instance  $x$  gets a probability of  $\frac{\alpha^{n_r(x)} D(x)}{L_r}$  in the new distribution  $D_r$ . Consider the case of two instances  $x_1$  and  $x_2$ , after  $(k - 1)$  iterations (just before learning the last hypothesis), when the first is correctly classified, and the second is incorrectly classified, by all the  $(k - 1)$  hypotheses. Regardless of the size of  $L_r$ , the quotient of the probabilities of the two instances would be  $\alpha^{(k-1)} = \alpha^{\text{poly}}$ , which is obviously exponential, thus implying that  $D_r$  is exponentially *far* from Uniform! Therefore, we must “correct” these intermediate distributions somehow, in-order to remain polynomially near-uniform. Nevertheless, we must ensure that these corrections would still allow us an error  $\leq \epsilon$ .

**The algorithm** As we shall soon see, in our DNF-learning algorithm we are not going to prove that the distributions are near-uniform and then use Theorem 2. Instead, we will almost repeat the proof we have shown in Theorem 2, and prove directly that for the distributions created in our algorithm we can weakly-learn the target function. Yet, since we still want our distributions to be polynomially near-uniform, then our goal remains the same: to ensure that for any instance  $x$ , at any stage of the algorithm, the number of correctly classifying hypotheses  $n_r(x)$ , would always remain high. Specifically, we would like it to be above  $(\lfloor \frac{r}{2} \rfloor - \Delta)$ , for some  $\Delta$  (to be chosen later). This will be done by defining, at the end of the  $r^{\text{th}}$

iteration, after learning  $h_r$ :

$$\hat{h}_r = \begin{cases} f & x \in X_{\lfloor \frac{r}{2} \rfloor - \Delta}^{(r)} \\ h_r & \text{otherwise} \end{cases}$$

We shall assume for now that we can define such hypotheses, although this is obviously impossible since we do not have  $f$  in hand. Later we will prove that the weight of the instances in  $X_{\lfloor \frac{r}{2} \rfloor - \Delta}^{(r)}$  is small, and thus, using such fictitious hypotheses is good enough for our needs.

This transformed hypothesis is the one we add to the set  $\hat{H}_{r+1}$ , and according to this new  $\hat{H}_{r+1}$  the  $n_r(x)$  values are then calculated. The intermediate  $D_{r+1}$  distribution is set according to  $\hat{H}_{r+1}$ , which is where this transformation actually affects our algorithm, since the consecutive hypothesis is learned over this distribution. At the end of the run, the *Majority* function is applied upon the “real”  $H_k$  set of hypotheses, to produce the algorithm’s output.

**Remark** Observe that we do not assume we have  $f$  only for those instances that are misclassified by *exactly*  $\lfloor \frac{r}{2} \rfloor - \Delta$  hypotheses, but we take this assumption for any number of misclassifying hypotheses  $\leq \lfloor \frac{r}{2} \rfloor - \Delta$ . This is true since once we get to the  $r^{\text{th}}$  phase for  $r = 2\Delta$ , then we assume for any instance in  $X_0^{(r)}$  that  $\hat{h}_r = f$  and thus, from now on, there is no instance in  $X_0^{(j)}$  for any  $j$ . Two iterations afterwards, we do the same thing to all the instances in  $X_1^{(r+2)}$ , while  $X_0^{(r+2)}$  is still an empty set, and so on. At each two additional iterations we add another set of instances to the ones we “throw away”, meaning we allow ourselves to be mistaken on these instances (in  $H_r$ , but not in the hypothetical  $\hat{H}_r$  which we analyze as if we indeed have  $f$  for our purposes). The immediate implication is that, as can be seen in figures 4.1 and 4.2, for any instance  $x \in X$  we have  $n_r(x) \geq \lfloor \frac{r}{2} \rfloor - \Delta$ .

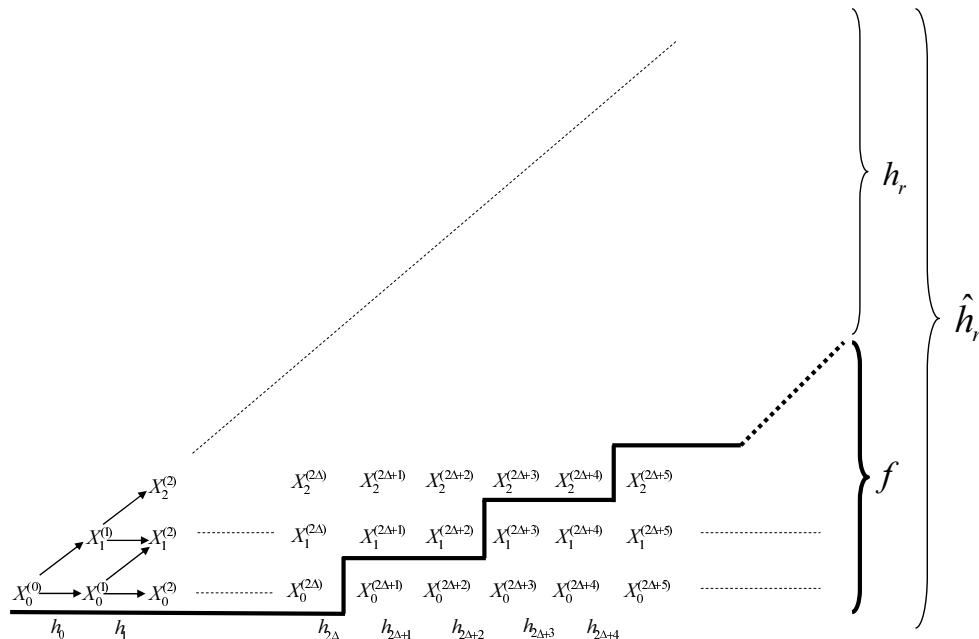


Figure 4.1: The instances “thrown away” through the algorithm’s progress.

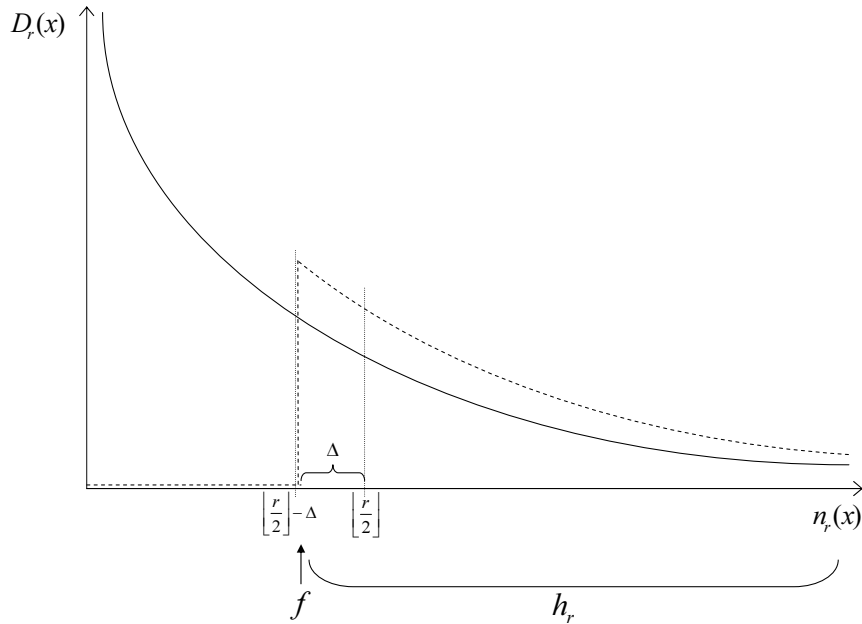


Figure 4.2: The distribution  $D_r$  before (straight) and after (dashed) “throwing” instances with low  $n_r(x)$ .

We shall now prove that for

$$\Delta = \frac{1 + 2\gamma}{4\gamma} \left( \ln k + \ln \frac{1}{\epsilon} \right)$$

$Maj(\hat{H}_k)$  “gets close enough” to  $f$  (for the same  $k$  defined in **AdaBoost**). Later we prove that outputting  $Maj(H_k)$  instead, do not add too much error to that of  $Maj(\hat{H}_k)$ , which gives us a way to efficiently approximate  $f$ .

**Lemma 7**  $Pr_D [f \neq Maj(\hat{H}_k)] \leq \epsilon$

**Proof:** Since the initial distribution is uniform, we have

$$\forall x \in X, D(x) = \frac{1}{2^n}$$

and at the  $r^{th}$  iteration

$$\forall x \in X, D_r(x) = \frac{\alpha^{n_r(x)}}{2^n L_r}$$

We should find a weak hypothesis to  $f$ , i.e. find the  $\chi_a$  (that is known to exist) such that

$$|E_{D_r} [f(x)\chi_a(x)]| \geq \frac{1}{4s}$$

writing the above expectation explicitly we get

$$\begin{aligned}
|E_{D_r} [f(x)\chi_a(x)]| &= \left| \sum_x D_r(x) f(x) \chi_a(x) \right| \\
&= \left| \sum_x \frac{\alpha^{n_r(x)} f(x)}{2^n L_r} \chi_a(x) \right| \\
&= \left| \sum_x \frac{1}{2^n} \cdot \frac{\alpha^{n_r(x)} f(x)}{L_r} \chi_a(x) \right| \\
&= \left| E_U \left[ \underbrace{\frac{\alpha^{n_r(x)} f(x)}{L_r}}_{g(x)} \chi_a(x) \right] \right| \geq \frac{1}{4s}
\end{aligned}$$

Note that the last expectation is over the uniform distribution, which means that if we see the under-braced part as a new function  $g(x)$ , then the whole expression can be rewritten as  $|\hat{g}_a| \geq \frac{1}{4s}$ . It would have been great if we could find this fourier-coefficient using [KM]. The “only” problem is that we can not estimate  $L_r$ , and therefore we do not know if  $|g(x)|$  is polynomially bounded (which is a prerequisite for using [KM]). We shall now see that we should not know  $L_r$  in-order to find this  $\hat{g}_a$ . We achieve this by taking constant parts of  $g(x)$  (that are not a function of  $x$ ) out of the expectation

$$\begin{aligned}
&\frac{\alpha^{\lfloor \frac{r}{2} \rfloor}}{L_r} \cdot \left| E_U \left[ \alpha^{n_r(x) - \lfloor \frac{r}{2} \rfloor} f(x) \chi_a(x) \right] \right| \geq \frac{1}{4s} \\
\Rightarrow \left| E_U \left[ \underbrace{\alpha^{n_r(x) - \lfloor \frac{r}{2} \rfloor} f(x)}_{g'(x)} \chi_a(x) \right] \right| &\geq \frac{1}{4s} \cdot \frac{L_r}{\alpha^{\lfloor \frac{r}{2} \rfloor}} \geq \frac{\epsilon_r}{4s} \geq \frac{\epsilon}{4s} = \frac{1}{\text{poly}(s, \frac{1}{\epsilon})} \quad (4.1)
\end{aligned}$$

Let  $g'(x)$  be the function under-braced above. Then the whole expression is the fourier-coefficient  $|\hat{g}'_a|$ . Note that there might be some  $\chi_a$  that satisfies the lower-bound for  $|\hat{g}'_a|$  but not for  $|\hat{g}_a|$ , meaning we can not just use [KM] to find such a  $\chi_a$  for  $|\hat{g}'_a|$ . Nevertheless, the  $\chi_a$  that *maximizes*  $|\hat{g}'_a|$  necessarily maximizes also  $|\hat{g}_a|$ , since we have only taken a constant out of the expectation. Thus, we may use [KM] to find the *maximal*  $|\hat{g}'_a|$ , which is greater-than  $\frac{1}{\text{poly}(s, \frac{1}{\epsilon})}$ , and by this we also find the maximal  $|\hat{g}_a|$  that, as we have seen, is also greater-than  $\frac{1}{\text{poly}(s, \frac{1}{\epsilon})}$ , as required.

What we should still prove, in-order to be able to use [KM], is that  $|g'(x)|$  is polynomially bounded.

**Claim 8**  $\forall x \in X, |g'(x)| \leq \text{poly}\left(\frac{1}{\epsilon}, s\right)$

**Proof:** By definition of  $g'(x)$ , and the observation that  $n_r(x) \geq \lfloor \frac{r}{2} \rfloor - \Delta$ :

$$|g'(x)| = \left| \alpha^{n_r(x) - \lfloor \frac{r}{2} \rfloor} f(x) \right| \leq \alpha^{n_r(x) - \lfloor \frac{r}{2} \rfloor} \leq \alpha^{-\Delta}$$

and by substituting  $\Delta$  we get

$$|g'(x)| \leq \left(1 - \frac{4\gamma}{1 + 2\gamma}\right)^{-\frac{1+2\gamma}{4\gamma} \ln \frac{k}{\epsilon}}$$

since  $(1 + \frac{x}{n})^n \leq e^x$ , we have

$$|g'(x)| \leq \frac{k}{\epsilon} = \frac{1}{2\gamma^2\epsilon} \ln \frac{1}{\epsilon} = \frac{1}{2(\frac{1}{4s})^2} \ln \frac{1}{\epsilon} = \frac{8s^2}{\epsilon} \ln \frac{1}{\epsilon} = \text{poly} \left( \frac{1}{\epsilon}, s \right)$$

as desired. ■

Therefore we can use [KM] and weak-learn  $f$  over the near-uniform distributions  $D_r$ , thus getting from **AdaBoost** a set of hypotheses such that

$$\Pr \left[ f \neq \text{Maj}(\hat{H}_k) \right] \leq \epsilon$$

which completes the proof of lemma 7. ■

We shall now prove that the additional error we get from ignoring some of the instances (the ones with less than  $\lfloor \frac{r}{2} \rfloor - \Delta$  correct hypotheses) is small, which means that the error of  $\text{Maj}(H_k)$  as the output hypothesis (instead of  $\text{Maj}(\hat{H}_k)$ ) is also small, as desired.

More specifically, we have proven above that  $\Pr \left[ f \neq \text{Maj}(\hat{H}_k) \right] \leq \epsilon$ , and we shall now prove that  $\Pr \left[ \text{Maj}(\hat{H}_k) \neq \text{Maj}(H_k) \right] \leq \epsilon$ , from which we deduce that:  $\Pr [f \neq \text{Maj}(H_k)] \leq 2\epsilon$  (see figure 4.3 for a graphical demonstration of the proof steps).

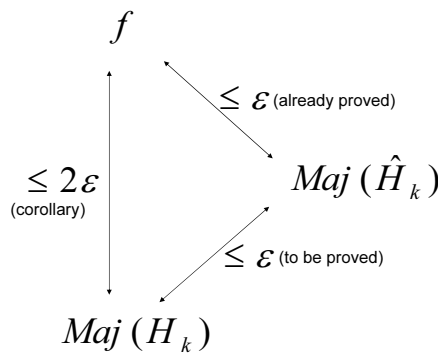


Figure 4.3: The “error-distances” between  $f$  and the hypotheses.

**Lemma 9**  $\Pr_D \left[ \text{Maj}(\hat{H}_k) \neq \text{Maj}(H_k) \right] \leq \epsilon$

**Proof:** Summing over all the sets of instances, for which we have “wrongly assumed” we have  $f$ , gives us an upper bound for the difference between  $\text{Maj}(\hat{H}_k)$  and  $\text{Maj}(H_k)$

$$\Pr_D \left[ \text{Maj}(\hat{H}_k) \neq \text{Maj}(H_k) \right] \leq \sum_{r=1}^k D \left( X_{\lfloor \frac{r}{2} \rfloor - \Delta}^{(r+1)} \right)$$

By the definition of  $D_r(x)$ :

$$\begin{aligned} D_r \left( X_{\lfloor \frac{r}{2} \rfloor - \Delta}^{(r+1)} \right) &= \frac{\alpha^{\lfloor \frac{r}{2} \rfloor - \Delta}}{L_r} D \left( X_{\lfloor \frac{r}{2} \rfloor - \Delta}^{(r+1)} \right) \\ \Rightarrow D \left( X_{\lfloor \frac{r}{2} \rfloor - \Delta}^{(r+1)} \right) &= \frac{L_r}{\alpha^{\lfloor \frac{r}{2} \rfloor - \Delta}} D_r \left( X_{\lfloor \frac{r}{2} \rfloor - \Delta}^{(r+1)} \right) \end{aligned} \quad (4.2)$$

and thus we get

$$Pr_D \left[ Maj(\hat{H}_k) \neq Maj(H_k) \right] \leq \sum_{r=1}^k \frac{L_r}{\alpha^{\lfloor \frac{r}{2} \rfloor - \Delta}} D_r \left( X_{\lfloor \frac{r}{2} \rfloor - \Delta}^{(r+1)} \right)$$

since  $D_r$  is a probability distribution (i.e. its values are  $\leq 1$ )

$$\leq k \cdot \frac{L_k}{\alpha^{\lfloor \frac{k}{2} \rfloor}} \cdot \alpha^\Delta$$

Since  $L_k \leq (1 - 2\gamma)^k$  (Claim 6), we get that  $L_k \leq \left( \frac{1-2\gamma}{1+2\gamma} \right)^{\lfloor \frac{k}{2} \rfloor} = \alpha^{\lfloor \frac{k}{2} \rfloor}$ , and so we can drop the  $\frac{L_k}{\alpha^{\lfloor \frac{k}{2} \rfloor}}$

$$\begin{aligned} &\leq k \cdot \alpha^\Delta \\ &= k \cdot \alpha^{\frac{1+2\gamma}{4\gamma} (\ln k + \ln \frac{1}{\epsilon})} \quad (\text{substitute } \Delta) \\ &= k \cdot \left( 1 - \frac{4\gamma}{1+2\gamma} \right)^{\frac{1+2\gamma}{4\gamma} \ln \frac{k}{\epsilon}} \quad (\text{substitute } \alpha) \end{aligned}$$

and (again) since  $(1 + \frac{x}{n})^n \leq e^x$ , we get:

$$Pr_D \left[ Maj(\hat{H}_k) \neq Maj(H_k) \right] \leq k \cdot \frac{\epsilon}{k} = \epsilon$$

As desired. ■

**Corollary 10**  $Pr_D [f \neq Maj(H_k)] \leq 2\epsilon$

Thus, by running the algorithm with  $\epsilon' = \frac{\epsilon}{2}$ , we get an  $\epsilon$ -approximation of the target function.

**Corollary 11** *DNF is PAC-learnable with membership-queries over the uniform distribution.*

For the sake of completeness, the final algorithm is detailed below. We denote by **KM-Max**( $t(x), \delta$ ) the algorithm that uses [KM] algorithm to find the *maximal* fourier-coefficient of the function  $t(x) \cdot f(x)$ , which has access to the membership oracle of  $f$  and can efficiently calculate  $t(x)$  (and thus can calculate  $t(x) \cdot f(x)$  efficiently). The algorithm detailed below receives  $\epsilon$  and  $\delta$  as its input:

- 1:  $\epsilon' = \frac{\epsilon}{2}$
- 2:  $k \leftarrow \frac{1}{2\gamma^2} \ln \frac{1}{\epsilon'}$
- 3:  $h_0 \leftarrow \mathbf{KM}(\delta)$
- 4:  $\hat{H}_1 \leftarrow \{h_0\}$
- 5:  $\hat{H}_1 \leftarrow \{h_0\}$
- 6: **For**  $r = 1, \dots, (k - 1)$  **do**

- 7:  $\forall x \in X, \quad n_r(x) \stackrel{\text{def}}{=} \left| \{ \hat{h} \in \hat{H}_r : \hat{h}(x) = f(x) \} \right|$
- 8:  $\forall x \in X, \quad t_r(x) \stackrel{\text{def}}{=} \alpha^{n_r(x) - \lfloor \frac{r}{2} \rfloor}$
- 9:  $h_r \leftarrow \mathbf{KM-Max}(t(x), \delta)$
- 10:  $\forall 0 \leq i \leq r, \quad X_i^{(r)} \stackrel{\text{def}}{=} \{x \in X | n_r(x) = i\}$
- 11:  $\forall x \in X, \quad \hat{h}_r(x) \stackrel{\text{def}}{=} \begin{cases} f(x) & x \in X_{\lfloor \frac{r}{2} \rfloor - \Delta}^{(r)} \\ h_r & \text{otherwise} \end{cases}$
- 12:  $H_{r+1} \leftarrow H_r \cup \{h_r\}$
- 13:  $\hat{H}_{r+1} \leftarrow \hat{H}_r \cup \{\hat{h}_r\}$
- 14: **end For**
- 15: **Output**  $Maj(H_k)$