

## Lecture 6: November 25, 2012

Lecturer: Yishay Mansour

Scribe: Uri Kanonov, Or Kamara, Ofir Hermesh<sup>1</sup>

## 6.1 Weak and Strong Learners

In the PAC model there is a distribution  $D$  on  $X$ . Random examples  $\langle x, c^*(x) \rangle$  are drawn according to the distribution  $D$  and the target function  $c^* \in \mathcal{C}$ . The goal is to find a hypothesis  $h \in H$  such that  $\text{error}(h, c^*) \leq \epsilon$ , with probability  $1 - \delta$ .

This is a *strong learning model*, since  $\epsilon$  and  $\delta$  can be arbitrarily small.

Recall that  $\epsilon$  is the error rate of the algorithm and  $1 - \delta$  represents the confidence. However, suppose we have an algorithm with low error rate but also low confidence, say confidence 50%, or alternatively an algorithm with an error rate of 49% (slightly better than flipping a coin) but high confidence level.

Is it possible to drive those *weak* algorithms to be *strong learners*? Intuitively, it is easier to find hypothesis that is correct only 51 percent of the time, rather than a hypothesis that is correct 99 percent of the time.

### 6.1.1 Boosting the confidence ( $1 - \delta$ )

Suppose algorithm  $A$  returns with probability  $\delta \geq \frac{1}{2}$  a hypothesis  $h$  such that  $\text{error}(h, c^*) \leq \epsilon$ . An interesting question is whether it is possible to build a PAC learning algorithm  $A'$  (from  $A$ )? The answer is Yes.

**Algorithm** *BoostConfidence*( $A$ ):

1. Run  $A$  for  $k = \log_{\frac{2}{\delta}} \frac{1}{\epsilon}$  times (on fresh data each time) with parameter  $\epsilon' = \frac{\epsilon}{2}$ . Algorithm  $A$  outputs hypothesis  $h_1 \dots h_k$ .
2. Draw a new sample  $S$  of size  $m = \frac{2}{\epsilon^2} \ln \frac{4k}{\delta} = O(\frac{1}{\epsilon^2} \ln \frac{k}{\delta})$  and test each hypothesis  $h_i$  on it, let  $\widehat{\text{error}}(h_i)$  be the observed error.  
Return  $\hat{h}^* = \min(\widehat{\text{error}}(h_i(S)))$ .

---

<sup>1</sup>Based on scribes written by Yoel Sharf, Livnat Jerby and Alon Ardenboim (November 21, 2010)

**Analysis of Algorithm** *BoostConfidence(A)*

After the first stage of the algorithm, with probability at most  $(\frac{1}{2})^k$ ,  $\forall i : error(h_i) > \frac{\epsilon}{2}$ . Hence, with probability at least  $1 - (\frac{1}{2})^k$ ,  $\exists i : error(h_i) \leq \frac{\epsilon}{2}$ .

Therefore, if we set  $k = \log_{\frac{1}{2}} \frac{2}{\delta}$ , we will get that with probability  $1 - \frac{\delta}{2}$  for one of  $h_1 \dots h_k$  holds  $error(h_i) \leq \frac{\epsilon}{2}$ .

Now we will show that after the second stage of the algorithm *BoostConfidence(A)*, with probability  $1 - \frac{\delta}{2}$ , outputs the hypothesis  $\hat{h}_i^*$  (with minimum errors on  $S$ ) such that,

$$error(\hat{h}_i^*) \leq \frac{\epsilon}{2} + \min(error(h_i)) \leq \epsilon \quad .$$

**Proof:** First, we use Chernoff Bound to bound the probability for “bad” event, i.e., the difference between the empirical error of any  $h_i$  and its real error is greater than  $\frac{\epsilon}{2}$ :

$$Pr[|\widehat{error}(h_i) - error(h_i)| \geq \frac{\epsilon}{2}] \leq 2e^{-2(\frac{\epsilon}{2})^2 m}$$

Second, we will bound by  $\frac{\delta}{2}$  the probability that such bad event will happen to any of the  $k$  hypothesis  $h_i$  using a Union Bound:

$$2ke^{-(\frac{\epsilon}{2})^2 m} \leq \frac{\delta}{2} \quad .$$

Then, by isolating  $m$ , we will get:

$$\begin{aligned} \frac{1}{e^{\frac{\epsilon^2}{2} m}} &\leq \frac{\delta}{4k} \\ \frac{4k}{\delta} &\leq e^{\frac{\epsilon^2}{2} m} \\ \ln \frac{4k}{\delta} &\leq \frac{\epsilon^2}{2} m \\ \frac{2}{\epsilon^2} \ln \frac{4k}{\delta} &\leq m \quad . \end{aligned}$$

We have that for a sample of size at least  $m$ , with probability  $1 - \frac{\delta}{2}$ , for each of those  $h_i$ :

$$\widehat{error}(h_i) - error(h_i) < \frac{\epsilon}{2}$$

thus

$$error(\hat{h}^*) - \min(error(h_i)) < \frac{\epsilon}{2} \quad .$$

From the first stage of the algorithm we already know that  $\min(error(h_i)) \leq \frac{\epsilon}{2}$ , hence:

$$error(\hat{h}^*) < \frac{\epsilon}{2} + \min(error(h_i)) \leq \epsilon$$

□

### 6.1.2 Boosting the accuracy ( $\epsilon$ )

One question we can ask: given an algorithm that outputs hypothesis with  $\epsilon = \frac{1}{2}$ , can we drive it to learn PAC? The answer is No, because such an algorithm will do exactly like flipping a coin.

#### Definition: Weak learning

Algorithm  $A$  learn Weak-PAC a concept class  $C$  with  $H$  if:

$\exists \gamma > 0$ ,

$\forall c^* \in C$ , (target function)

$\forall D$ , (distribution)

$\forall \delta > \frac{1}{2}$ ,

Algorithm  $A$  outputs hypothesis  $h \in H$  and with probability  $1 - \delta$ , such that  $error(h) \leq \frac{1}{2} - \gamma$ .

Intuitively,  $A$  will guarantee an error rate of 49% instead of 1% of the PAC model. We show, that if a concept class has a weak learning algorithm, then there is a PAC learning algorithm for the class.

Note that running  $A$  multiple times on the same distribution  $D$ , does not work because  $A$  might return the same hypothesis over and over again.

#### Example

Suppose we have the following target function  $c^*$  (over bits) with a Uniform distribution  $D$ :

if  $x_1 = x_2 = 1 \implies c^*(x) = \text{some very hard function}$

otherwise  $\implies c^*(x) = 0$

(e.g., the hardness depends on the first and the second bits.)

We can easily achieve 87.5% accuracy by flipping a coin if  $x_1 = x_2 = 1$  and otherwise predicting zero.

The probability for the event  $x_1 = x_2 = 1$  is 0.25 which gives us a total accuracy of 87.5%. On the other hand, getting better than 87.5% accuracy is hard. For this reason we want our weak learner to perform well with any distribution  $D$ ! (In the example a natural distribution is  $x_1 = x_2 = 1$  and uniform otherwise).

Conclusion: An important requirement in weak learning model is: *for all distribution*. (in the example we assumed a specific distribution)

## Proof of Equivalence between Weak and Strong Learning

Preliminary: We are going to concentrate on the case of a sample with finite size, so that, given  $x_1 \dots x_m$  goal is to find a hypothesis classifies all of them correctly.

Given:  $x_1 \dots x_m$  and their labels,

$H$  - A weak hypothesis class,

We will use a Regret Minimization algorithm RM.

The Algorithm:

1. The algorithm basic action: to choose a distribution over  $x_1 \dots x_m$ .
2. For each step  $t$ , the RM(Regret Minimization) algorithm chooses a distribution (over  $x_1 \dots x_m$ ).
3. For each step  $t$ , the opponent returns loss (for each  $x_i$ ) such that,  $error(h_t, D_t) \leq \frac{1}{2} - \gamma$ . The loss to be returned will be 1 for correct classification and 0 for error (opposite of what we expect).
4. After  $T$  steps, we will require that  $MAJ(h_1(x) \dots h_T(x))$  classify all the sample correctly.

Notes:

1. Since at each stage we return a weak-learner  $h$ , that is classified correctly at least  $\frac{1}{2} + \gamma$  fraction, then the loss of RM is at least  $(\frac{1}{2} + \gamma)T$ .
2. Suppose that there is  $x_i$  such that MAJ does not classify it correctly, then the loss of  $x_i$  is at most  $\frac{T}{2}$ .
3.  $loss(RM) = (\frac{1}{2} + \gamma)T \leq \frac{T}{2} + 2\sqrt{T \log m}$ .  
( $2\sqrt{T \log m}$  is the regret bound of RM)  
For this inequality to hold, it is required that:

$$\gamma T \leq 2\sqrt{T \log m}$$

$$T \leq \frac{4 \log m}{\gamma^2}$$

4. Conclusion: If you execute the above RM algorithm, after  $\frac{4 \log m}{\gamma^2}$  iterations you will get a consistent hypothesis.

Thus, by *Occam Razor Theorem*, we can PAC learn the class.

5. Remark: RM will naturally concentrate on examples with the ‘most errors’.

## 6.2 Recursive Construction

### 6.2.1 Algorithm Description

Let  $A$  be a weak learning algorithm, and  $p$  the error probability of  $A$ .

**Step 1:** Run  $A$  with the initial distribution  $D_1$  to obtain  $h_1$  (*error*  $\leq \frac{1}{2} - \gamma$ ).

**Step 2:** Define a new distribution  $D_2$ , such that

$$\begin{aligned} S_c &= \{x | h_1(x) = c^*(x)\} \\ S_e &= \{x | h_1(x) \neq c^*(x)\} \\ D_2(S_c) &= D_2(S_e) = \frac{1}{2} \end{aligned}$$

To do so we will define  $D_2$  as follows:

$$D_2(x) = \begin{cases} \frac{0.5}{1-p} \cdot D_1(x) & x \in S_c \\ \frac{0.5}{p} \cdot D_1(x) & x \in S_e, \end{cases}$$

where  $p = D_1(S_e)$ .

To obtain  $h_2$  we will run  $A$  with  $D_2$ .

**Step 3:** The distribution  $D_3$  would be defined only on examples  $x$  for which  $h_1(x) \neq h_2(x)$ :

$$D_3(x) = \begin{cases} \frac{D_1(x)}{Z} & h_1(x) \neq h_2(x) \\ 0 & \text{otherwise,} \end{cases}$$

where  $Z = P[h_1(x) \neq h_2(x)]$ .

Our combined hypothesis would be:

$$H(x) = \begin{cases} h_1(x) & h_1(x) = h_2(x) \\ h_3(x) & \text{otherwise} \end{cases}$$

Which is equivalent to  $H(x) = MAJ(h_1(x), h_2(x), h_3(x))$ .

## 6.2.2 Estimation of the Error

Suppose each hypothesis  $h_i$  errors with a probability of  $p$ , independently. What would be the error of the majority of  $h_1, h_2, h_3$ ?

$$\text{Error} = 3p^2(1-p) + p^3 = 3p^2 - 2p^3 = p^2(3-2p)$$

We would like to show that this is the error probability without assuming the hypotheses are independent. To do so we would partition the space into four subspaces:

$$\begin{aligned} S_{cc} &= \{x | h_1(x) = c^*(x) \wedge h_2(x) = c^*(x)\} \\ S_{ee} &= \{x | h_1(x) \neq c^*(x) \wedge h_2(x) \neq c^*(x)\} \\ S_{ce} &= \{x | h_1(x) \neq c^*(x) \wedge h_2(x) = c^*(x)\} \\ S_{ec} &= \{x | h_1(x) = c^*(x) \wedge h_2(x) \neq c^*(x)\} \end{aligned}$$

Let  $P_{cc} = D_1(S_{cc})$ ,  $P_{ee} = D_1(S_{ee})$ ,  $P_{ce} = D_1(S_{ce})$  and  $P_{ec} = D_1(S_{ec})$ .

The error probability, with respect to the initial distribution  $D_1$ , is  $P_{ee} + (P_{ec} + P_{ce})p$ .

Let us define  $\alpha = D_2(S_{ce})$ . Therefore, from the definition of  $D_2$ , in terms of  $D_1$  we get  $P_{ce} = 2(1-p)\alpha$ .

Since  $D_2(S_{*e}) = p$ , we have,

$$\begin{aligned} D_2(S_{ee}) &= p - \alpha \\ P_{ee} &= 2p(p - \alpha). \end{aligned}$$

From the construction of  $D_2$ ,

$$\begin{aligned} D_2(S_{ec}) &= \frac{1}{2} - (p - \alpha) \\ P_{ec} &= 2p\left(\frac{1}{2} - p + \alpha\right). \end{aligned}$$

Therefore the error is:

$$P_{ee} + (P_{ec} + P_{ce})p = 2p(p - \alpha) + p(2p(\frac{1}{2} - p + \alpha) + 2(1-p)\alpha) = 3p^2 - 2p^3.$$

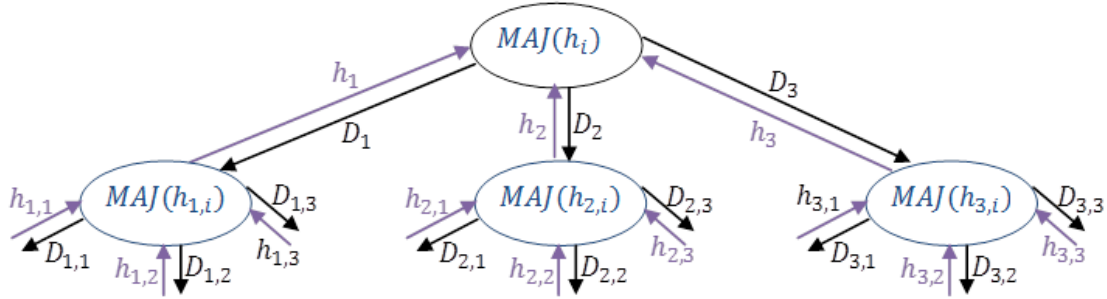
Given that the initial probability is  $p_0 = \frac{1}{2} - \gamma_0$ , each step improves upon the previous step:

$$\begin{aligned} \frac{1}{2} - \gamma_{t+1} &= 3\left(\frac{1}{2} - \gamma_t\right)^2 - 2\left(\frac{1}{2} - \gamma_t\right)^3 \\ \frac{1}{2} - \gamma_{t+1} &\leq \frac{1}{2} - \gamma_t\left(\frac{3}{2} - \gamma_t\right) \end{aligned}$$

The termination condition of the recursion would be to obtain an error of  $\epsilon$  as required. For  $\gamma_t > \frac{1}{4}$  we get that  $p < \frac{1}{4}$ , and therefore

$$p_{t+1} \leq 3p_t^2 - 2p_t^3 \leq 2p_t^2 < \frac{1}{2}$$

The recursion depth is therefore  $O(\log \frac{1}{\gamma} + \log(\log \frac{1}{\epsilon}))$ , and the number of nodes in the recursion tree is  $3^{\text{depth}} = \text{poly}(\frac{1}{\gamma}, \log \frac{1}{\epsilon})$ .  $\square$



## 6.3 AdaBoost

The AdaBoost algorithm is an iterative boosting algorithm that enables us to create a strong learning algorithm from a weak learning algorithm. The general idea of this algorithm is to maintain a distribution on the input sample, and increase the weight of the harder to classify examples so the algorithm would focus on them.

### 6.3.1 Algorithm Description

**Input:** A set of  $m$  classified examples:  $S = \{ \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \dots, \langle x_m, y_m \rangle \}$  where  $\forall i \in \{1, \dots, m\} \times y_i \in \{-1, 1\}$ .

**Definitons:** Let  $D_t$  denote the distribution of weights of the examples at time  $t$ , and  $D_t(i)$  the weight of example  $x_i$  at time  $t$ .

**Initialization:**

$$D_1(i) = \frac{1}{m} \quad \forall i \in \{1, \dots, m\}$$

**Step:** At each iteration we use a classifier  $h_t \in \mathcal{H}: X \mapsto \{-1, +1\}$  that minimizes the error on the current distribution (defined as  $\epsilon_t = \Pr_{D_t}[h_t(x) \neq c_*(x)]$  where  $c_*$  is the target function).

At time  $t + 1$  we update the weights in the following manner:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & y_i = h_t(x_i) \\ e^{\alpha_t} & y_i \neq h_t(x_i) \end{cases} \\ &= \frac{D_t(i)}{Z_t} \cdot e^{-y_i \alpha_t h_t(x_i)} \end{aligned}$$

where  $Z_t$  is a normalizing factor to keep  $D_{t+1}$  a distribution and  $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ .

**Output:** The hypothesis we return after running the algorithm for  $T$  iterations is:

$$H(x) = \text{Sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right)$$

An advantage using the AdaBoost algorithm is that it removes the need of knowing the parameter  $\gamma$ . Another advantage is that it's easy to implement and runs efficiently.

### 6.3.2 Bounding the Error

**Theorem 6.1** *Let  $H$  be the output hypothesis of AdaBoost. Then:*

$$\begin{aligned} \widehat{\text{error}}(H) &\leq \prod_{t=1}^T 2\sqrt{\epsilon_t(1-\epsilon_t)} \\ &= \prod_{t=1}^T \sqrt{1-4\gamma_t^2} \\ &\leq e^{-2\sum_t \gamma_t^2} \end{aligned}$$

where the last line is obtained from the inequality  $1 + x \leq e^x$ .

**Conclusion:** the error drops exponentially fast in  $T$ .

**Proof:** The proof follows in three steps:

1. First, obtain the following expression for  $D_{T+1}(i)$ :

$$D_{T+1}(i) = \frac{D_1(i) e^{-y_i f(x_i)}}{\prod_t Z_t}$$

where  $f(x) = \sum_{t=1}^T \alpha_t h_t(x)$ .

**Proof:** Since  $D_{t+1}(i)$  is given by:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} e^{-y_i \alpha_t h_t(x_i)}$$



we can unravel the recurrence to obtain:

$$\begin{aligned}
 D_{T+1}(i) &= D_1(i) \prod_{t=1}^T \frac{e^{-y_i \alpha_t h_t(x_i)}}{Z_t} \\
 &= D_1(i) \frac{e^{-y_i \sum_{t=1}^T \alpha_t h_t(x_i)}}{\prod_{t=1}^T Z_t} \\
 &= D_1(i) \frac{e^{-y_i f(x_i)}}{\prod_t Z_t}
 \end{aligned}$$

□

2. Second, we bound the training error of  $H$  by the product of the normalizing factors  $Z_t$ :

$$\widehat{error}(H) \leq \prod_{t=1}^T Z_t$$

**Proof:**

$$\begin{aligned}
 \widehat{error}(H) &= \frac{1}{m} \sum_{i=1}^m I(y_i \neq H(x_i)) \\
 &= \frac{1}{m} \sum_{i=1}^m I(y_i f(x_i) \leq 0) \\
 &\leq \frac{1}{m} \sum_{i=1}^m e^{-y_i f(x_i)} \\
 &= \frac{1}{m} \sum_{i=1}^m m \left( \prod_{t=1}^T Z_t \right) D_{T+1}(i) \\
 &= \left( \prod_{t=1}^T Z_t \right) \sum_{i=1}^m D_{T+1}(i) \\
 &= \prod_{t=1}^T Z_t,
 \end{aligned}$$

where  $I$  is the indicator function. The third line follows from the observation that when  $I(y_i f(x_i) \leq 0) = 1$ , then  $y_i f(x_i) \leq 0$  and so  $e^{-y_i f(x_i)} \geq 1 = I(y_i f(x_i) \leq 0)$ . (Also, clearly when  $I(y_i f(x_i) \leq 0) = 0$ , then  $e^{-y_i f(x_i)} \geq 0$ ). The fourth line follows from step 1. The last line is obtained from the fact that  $D_{T+1}$  is a probability distribution over the examples.

□

3. Now that the training error has been bounded in step 2 by the product of the normalizing weights  $Z_t$ , the last step is to express  $Z_t$  in terms of  $\epsilon_t$ :

$$Z_t = 2\sqrt{\epsilon_t(1 - \epsilon_t)}$$

**Proof:** By definition,

$$\begin{aligned} Z_t &= \sum_{i=1}^m D_t(i) e^{-y_i \alpha_t h_t(x_i)} \\ &= \sum_{i: y_i = h_t(x_i)} D_t(i) e^{-\alpha_t} + \sum_{i: y_i \neq h_t(x_i)} D_t(i) e^{\alpha_t} \\ &= (1 - \epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t}, \end{aligned}$$

where the last step follows from the definition of  $\epsilon_t$ :

$$\sum_{i: y_i \neq h_t(x_i)} D_t(i) = \epsilon_t,$$

Since the expression above for  $Z_t$  is valid for all  $\alpha_t$ , minimizing  $Z_t$  with respect to  $\alpha_t$  for each  $t$  will produce the minimum training error  $\widehat{error}(H)$ .

$$\frac{\partial Z_t}{\partial \alpha_t} = -(1 - \epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t} = 0$$

Solving, we find:

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right).$$

□

Using this value of  $\alpha_t$  in the expression for  $Z_t$ , and then plugging that into the bound on the training error for  $H$ , we end up with:

$$\widehat{error}(H) \leq \prod_{t=1}^T \left( 2\sqrt{\epsilon_t(1 - \epsilon_t)} \right)$$

which proves the theorem. □