Computational Learning Theory	Spring semester, 2002

Lecture 4: May 18, 2002

Lecturer: Yishay Mansour *Scribe: Tamir Berler and Ron Moshe Hecht* [1]

4.1 Experts and Boosting Algorithms

4.1.1 Motivation

Consider a situation in which we have several experts for a certain task. However we don't have any prior information on the reliability of the different experts. We want to use the experts guesses to predict, and hope to converge to the performance of the best expert. (Of course we can not expect to perform better than the best expert, when dealing with the general case.). For example assume that we have accesses to a set of weather forecasts and we want to the predict the weather of the following day. based on the prediction of them.

4.1.2 The online model

In each step we have full information about the past, and no information about the future. (In each step we know exactly whether an expert was correct in his guess, or was wrong.)

The model includes:

1. N different strategies (experts).
2. In step t, the learner A, chooses a distribution P^t on the N strategies. ($\sum_{i=1}^{N} P^t(i) = 1$)
3. After each step, the learner receive the loss of each expert (denote by the vector l^t).
4. The loss of A at time t is $\sum_{i=1}^{N} P^t(i) l_i^t$.

Without the loss of generality we can assume that $l_i^t \in [0,1]$. Our goal is to build an algorithm A that minimize the loss of the learner. Of course the loss of A is at least the loss of the best expert. The loss of the i-th expert is given by $L_i^t := \sum_{j=1}^{t} l_i^j$.

[1] Based on scribes written by Ofir Friedman (March 25,1997)

4.1.3 Examples

Guessing letters

We have k letters and k experts, where expert i guess the i-th letter all the time. The loss is 0 if our letter matches the next letter, and 1 if it does not. It is easy to see, that in the worst case we will loses n in n successive trials. However, the best expert loses no more then $\frac{n}{k}$ of the trails. The idea here is to guess the most popular letter online, Since the expert predicting the most popular letter is the expert with the best performance . We would like to perform as the best expert does.

player 1 \ player2	Rock	Paper	Scissors
Rock	0.5	1	0
Paper	0	0.5	1
Scissors	1	0	0.5

Figure 4.1: 2 players game

Game theory

Two players play a game. player 1 chooses a row, player 2 chooses a column. Each of the players have 3 different strategies (Experts). The loss of player 1, is as shown on the table above.
Given that player 2 chooses his strategies in random (nature), no matter what will be the strategy of player 1, he will loose $\frac{n}{2}$ in n games.
If, on the other hand, player 2 has one fixed strategy, then player 1 must choose his best

corresponding strategy.

Placing a point

One needs to choose a point d on a plain such that it is as close as possible to the target y. The loss function is expressed as the distance between the d and y $L = |d - y|$. If the loss function is a convex function i.e. $|(\lambda d_1 + (1-\lambda)d_2) - y| \leq \lambda(|d_1 - y|) + (1-\lambda)(|d_2 - y|)$ then the best estimation for y can be expressed as the center of all the experts.

4.1.4 First Experts Algorithm: The Greedy Algorithm

We first introduce the following definition.
Definition: L_i is the cumulative loss of the i^{th} expert that is expressed as follows: $L_i^t := \sum_{j=1}^{t} l_i^j$. where L_i^j is the loss of the i^{th} expert in the j^{th} step
and t is the time of the last prediction.
Greedy algorithm: choose the expert i that achieved the minimum loss till time t, i.e.
the algorithm decision at step $t + 1 is arg \min_i L_i^t$

Theorem 4.1 *Let L_G^T be the loss of the greedy algorithm at time T. Then:*

$$L_G^T \leq N(\min_i L_i^T + 1)$$

where N is the number of experts.

Proof: Let G_i^t be the loss of the greedy algorithm that was incurred when the algorithm choose the expert i.

$$G_i^t = \begin{cases} G_i^{t-1} + l_i^t & i = arg \min_k L_k^{t-1} \\ G_i^{t-1} & else \end{cases} \tag{4.1}$$

Clearly, $\forall i, l_i^t \leq L_i^t$. Now we are ready to show that:

$$\forall t, j \ G_j^t \leq \min_i (L_i^t) + 1$$

we consider three different cases regarding the last time the algorithm choose the j^{th} expert.

1. At time t i.e i=j
   $G_j^t = G_i^t \leq L_i^t < L_i^t + 1$

2. The algorithm never has chosen the j^{th} expert
   $G_j^t = 0 \leq L_i^t < L_i^t + 1$
3. The last time the algorithm choose the j^{th} expert was at time k. Notice that the j^{th} expert had the minimal loss at that time. Therefore:
   $\forall i, L_j^{t-k} \leq L_i^{t-k}$
   $G_j^t - k \leq L_j^{t-k} \leq L_i^{t-k} < L_i^{t-k} + 1$
   Since the j expert was not chosen for the last k time steps
   $G_j^t - k = G_j^t$
   $G_j^t = G_j^t - k \leq L_i^{t-k} + 1 \leq L_i^{t-k} + 1 + \sum_{n=t-k+1}^{t} l_n^i = L_i^t$

Thus:

$$L_G^T = \sum_{j=1}^{N} G_j^T \leq \sum_{j=1}^{N} \min_i(L_i^T) + 1 \leq N(\min_i(L_i^T) + 1)$$

■

From the guessing letters example, one can see that this bound is tight.

4.1.5 Second Experts Algorithm: The Hedge Algorithm

Next we present a much better algorithm. In each step t, the algorithm maintains a positive weight vector w^t for the experts.
We define the probability vector $P^t = \frac{w^t}{\sum_{i=1}^{N} w_i^t}$.
We initialize for every i $w_i^1 = \frac{1}{N}$. After receiving l^t, we update the weights as follows:

$$w_i^{t+1} = w_i^t U_\beta(l_i^t)$$

Where $U_\beta : [0,1] \rightarrow [0,1]$, such that for any $r, \beta \in [0,1]$, we have

$$\beta^r \leq U_\beta(r) \leq 1 - (1-\beta)r.$$

Let us try to motivate the update rule. Assume that $U_\beta(r) = \beta^r$, then:

$$w_i^t = w_i^{t-1}\beta^{l_i} = \ldots = w_i^1 \beta^{L_i^{t-1}}.$$

Note, that when an expert is wrong, his weight decreased, and he will have a smaller effect on the following guesses. If we have one (or several) good expert, his weight in the final guess will dominate, and the performance of the algorithm will be close to that best expert. Unlike the previous algorithm, the predictions will not be dominated by a single expert at each step but by the set of almost bests experts.

Analysis of Hedge

Lemma 4.2 *For any sequence of loses, $l^1 \dots l^T$,*

$$ln(\sum_{i=1}^{N} w_i^{T+1}) \leq -(1-\beta)L_{Hedge(\beta)}$$

Proof: Choose a certain time t, then we obtain,

$$\sum_{i=1}^{N} w_i^{t+1} = \sum_{i=1}^{N} w_i^t U_\beta(l_i^t) \leq \sum w_i^t(1-(1-\beta)l_i^t) = (\sum_{j=1}^{N} w_j^t)(1-(1-\beta)P^t l^t)$$

(For the last equality we use the identity $w_i^t = P_i^t(\sum w_j^t)$).

We will take the log of the expression, and using the inequality $ln(1-x) \leq -x$, we have:

$$ln(\sum w_i^{t+1}) \leq ln(\sum w_j^t) + ln(1-(1-\beta)P^t l^t) \leq ln(\sum w_i^t) - (1-\beta)P^t l^t$$

Hence, the growth of the log of the weight in each step, is bounded by $-(1-\beta)P^t l^t$. Therefore,

$$ln(\sum w_i^{t+1}) \leq ln(\sum w_i^1) - (1-\beta)\sum P^t l^t = -(1-\beta)L_{Hedge(\beta)}$$

■

The Lemma implies :

$$L_{Hedge(\beta)} \leq \frac{-ln(\sum_{i=1}^{N} w_i^{t+1})}{1-\beta}$$

Since the weights are positive for any i, we get:

$$\sum w_i^{T+1} \geq w_j^{T+1} \geq w_j^1 \beta^{L_j} = \frac{1}{N}\beta^{L_j}$$

Therefore:

$$\forall i \quad L_{Hedge} \leq \frac{-ln(\frac{1}{N}\beta^{L_i})}{1-\beta} = \frac{lnN + L_i ln\frac{1}{\beta}}{1-\beta}$$

Since the last equation holds for all of the experts, it in particularly holds for the best expert. (The best expert is the one whose loss is minimal.) Lets the best be the k expert. We have

$$L_{Hedge} \leq \frac{lnN + L_k ln\frac{1}{\beta}}{1-\beta}$$

Having bounded the loss of the hedge algorithm in terms of the loss of the best expert, all that is left is to set β that provides the best bound.

If we set $\beta = \frac{1}{2}$, we get the bound of $2lnN + (ln2)min_i(L_i)$, which is much better that what we had (A fairly weak dependency on the number of experts). We can choose β to get an even better bound.

Lemma 4.3 *For $0 \leq L \leq L'$ and $0 \leq R \leq R'$ if we choose $\beta = g(\frac{L'}{R'})$ where $g(z) = \frac{1}{1+\sqrt{\frac{2}{z}}}$ then*

$$\frac{-Lln\beta + R}{1-\beta} \leq L + \sqrt{2L'R'} + R$$

Using the above lemma, we can choose $L' = maximumlose$ and $R' = lnN$ and get that for $\beta = g(\frac{L'}{lnN}$

$$L^A \leq min(L_i) + \sqrt{2L'lnN} + lnN$$

4.2 Occam Razor

"*Entities should not be multiplied unnecessarily*"

(William of Occam, c. 1320)

The above quote is better known as Occam's Razor or the principle of Ontological Parsimony. Over the centuries, people from different fields have given it different interpretations. One interpretation used by experimental scientists is: given two explanations of the data, the simpler explanation is preferable. This principle is consistent with the goal of machine learning: to discover the simplest hypothesis that is consistent with the sample data. However, the question still remains: why should one assume that the simplest hypothesis based on past examples will perform well on future examples. After all, the real value of a hypothesis is in predicting the examples that it has not yet seen. It will be shown that, under very general assumptions, Occam's Algorithm produces hypotheses that with high probability will be predictive of future observations. As a consequence, when hypotheses of minimum or near minimum complexity can be produced in time which is polynomial in the size of the sample data, it leads to a polynomial PAC-learning algorithm.

Here, a simple hypothesis, means a short coded hypothesis. We saw that we can achieve a polynomial PAC-learning algorithm, for a finite class of hypotheses, H, if we choose the number of input examples m, to be:

$$m \geq \frac{1}{\epsilon} \ln \frac{|H|}{\delta}$$

We can allow H to grow with m, providing that it grows slower than m.
Definition: An (α, β) *Occam-algorithm* for a functions class C, using a hypotheses class H, is an algorithm which has two parameters: a constant $\alpha \geq 0$ and a compression factor $0 \leq \beta < 1$. Given a sample S of size m, which is labelled according to c_t, i.e. $\forall\ i \in \{1, \cdots, m\}\ \ c_t(x_i) = \sigma_i$, the algorithm outputs an hypothesis $h \in H$, such that:

1. The hypothesis h is consistent with the sample, i.e. $\forall\ \ i \in \{1, \cdots, m\}\ \ h(x_i) = \sigma_i$.
2. The size of h is at most $n^\alpha m^\beta$, where n is the size of c_t, m is the sample size, and α and β are the algorithm's parameters.

4.2.1 Occam Algorithm and Compression

An Occam algorithm gives a way to compress the sample. The n bits σ_i can be replaced by the $n^\alpha m^\beta$ bits that represent h. Thus, suppose A wants to transmit σ_i to B, and both share the x_i, it is enough for A to transmit h, because B would be able to compute the values of σ_i by evaluating $h(x_i) = \sigma_i$. Therefore we have the same information in a more compact form. The use of the Occam algorithm for compression can be one step farther, to deal with situation in which the classification is not perfect. When dealing with situation that an error might occur, one needs to transmit the error in addition to the model h.

4.2.2 Occam Algorithm and The PAC Learning model

We now show that the existence of an Occam-algorithm for $\mathcal{C}$ implies polynomial PAC Learning ability.

Theorem 4.4 *Let A be an (α, β) Occam Algorithm for C, using H. Let D be the distribution according to which the samples, X, are drawn. Let $c_t \in C$ be the target concept, and n be its size. For any $0 \leq \epsilon, \delta \leq 1$, if A gets as an input a sample, S, which has m examples drawn from $EX(c_t, D)$, where*

$$m \geq \left(\frac{n^\alpha}{\epsilon} \ln\left(\frac{2}{\delta}\right)\right)^{1/(1-\beta)}$$

then the probability that the output, $h \in H$, satisfies : $error(h) \leq \epsilon$, is at least $1 - \delta$.

Remark: Notice, that when $\beta \to 1$, then $m \to \infty$. This makes sense, since the closer β is to 1, the poorer the compression we get, and hence we have less "information" about the function.

Proof: Since A returns a hypothesis h, satisfying: $size(h) \leq n^\alpha m^\beta$, and assuming the hypotheses under consideration are given by binary strings, the size of the hypothesis space is no greater than $2^{n^\alpha m^\beta}$. We showed that if:

$$m \geq \frac{1}{\epsilon} \ln \frac{|H|}{\delta}$$

then any consistent $h \in H$ satisfies $error(h) \leq \epsilon$ with probability at least $1 - \delta$.
Since $|H| \leq 2^{n^\alpha m^\beta}$ we get the same conclusion for:

$$m \geq \frac{1}{\epsilon} \ln \frac{2^{n^\alpha m^\beta}}{\delta}$$

We can conclude that

$$m \geq \frac{n^\alpha m^\beta}{\epsilon} \ln \frac{2}{\delta}$$

We need to solve for m,

$$m^{\beta-1} \geq \frac{n^\alpha}{\epsilon} \ln \frac{2}{\delta}$$

therefore,

$$m \geq \left(\frac{n^\alpha}{\epsilon} \ln\left(\frac{2}{\delta}\right)\right)^{1/(1-\beta)}$$

which completes the proof. ■

4.3 Weak and Strong Learners

In the PAC model there is a distribution D on X. Random examples $< x, c(x) >$ are drawn according to D and $c \in C$ is the target function. The goal is to find a hypothesis $h \in H$ such that $error(h, c) \leq \epsilon$, with probability $1 - \delta$.
This is a *strong learning model*, since ϵ can be as small as we want.

We define a *weak learning model* by replacing the accuracy requirement by :

$$error(h, c) \leq \frac{1}{2} - \gamma$$

.

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. We show, that if a concept class has a weak learning algorithm, then there is a PAC learning algorithm for the class.
We will change the definition : $error(h, c) = E_{X\ D}[|h(x) - c(x)|]$, when $c(x) \in \{0,1\}$ and $h(x) \in [0, 1]$, meaning that the hypothesis is probabilistic, rather than deterministic.

4.3.1 The Majority Algorithm

First, let us define our prediction hypothesis. For any set of h function from H, we have a hypothesis which is their majority, i.e. $MAJ(h_1 \ldots h_T)$, for $h_i \in H$. Our PAC learning algorithm will work as follows.
We will take sample of m points, find a hypothesis $h^A = MAJ(h_1 \ldots h_T)$, which is consistent with the sample. If we can guarantee that,

$$size(h_A) = Tsize(h_i) << m,$$

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

We will use the weak hypothesis as follows. We will take the sample of m points with a uniform distribution and look for a weak hypothesis from H. If we find the weak hypothesis has error less than $\frac{1}{m}$, we are done, since it has to be consistent. Else, we change the distribution on the m points, and get another weak hypothesis h_i. At the end we return the majority of all the weak hypothesis h_i.

First, we will assume that we know a value γ, such that the errors $\epsilon_1 \ldots \epsilon_T$ are bounded by $\frac{1}{2} - \gamma$, where $\epsilon_i = error(h_i, c)$. The "experts" will be the points. The loss of the expert i in step t will be $l_i^t = 1 - |h_t(x_i) - c(x_i)|$ (Note that the loss is 1 if h_t is correct on x_i and 0 if it is incorrect). We fix $\beta = 1 - \gamma$ and our final hypothesis $h^A = MAJ(h_1 \ldots h_T)$. We need to show how the error of h^A drops when T increases.

Define, $S = \{i | h^A(x_i) \neq c(x_i)\}$, i.e. the set of points on which our final hypothesis errors. For $i \in S$

$$\frac{L_i}{T} = \frac{1}{T}\sum_{t=1}^{T} l_i^t = 1 - \frac{1}{T}\sum_{t=1}^{T} |c(x_i) - h_t(x_i)| = 1 - |c(x_i) - \frac{1}{T}\sum_{t=1}^{T} h_t(x_i)| \leq \frac{1}{2},$$

From what we saw before on $Hedge(\beta)$,

$$L_M = \sum P^t l^t \leq \frac{-ln(\sum_{i \in S} D(i)) - ln(\beta)L_i}{1-\beta} \leq \frac{-ln(\sum_{i \in S} D(i)) - ln(\beta)\frac{T}{2}}{\gamma}.$$

Since $-ln\beta = -ln(1-\gamma) \leq \gamma + \gamma^2$
by combining those two equations we get:

$$T(\frac{1}{2} + \gamma) \leq \sum P^t l^t \leq \frac{-ln(\sum_{i \in S} D(i)) + (\gamma + \gamma^2)\frac{T}{2}}{\gamma}.$$

The probability of error is $\epsilon = \sum_{i \in S} D(x_i)$ and thus:

$$\epsilon \leq e^{\frac{-T\gamma^2}{2}}$$

We need that $\epsilon \leq \frac{1}{m}$ and get: $T \geq \frac{2ln(m)}{\gamma^2}$

We achieved our goal. We showed that if a class has a weak learning algorithm, it has a strong learning algorithm. From a practical point of view it is very unfortunate that we need to know γ in advance, to run the algorithm (we need it in order to define β).

4.3.2 The AdaBoost Algorithm

The advantage of this algorithm is that we don't need to know γ in advance, and that we don't need a uniform bound on all the weak learners. In addition, we will be able to show better error bounds. AdaBoost in each iteration, uses a different β_t depend on ϵ_t, which is the error of the last weak hypothesis.
Consider the framework that we had before. The input of the algorithm is a set of m sample x_i that were drawn from the domain X. For each example we know its real classification $c(x_i)$. However this time we limit ourselves to a constant number of iterations T.
Initialization : $w_i^1 = D(x_i)$
For each step t The AdaBoost algorithm does the following:

1. calculate the probability that is assigned to each sample i $P_i^t = \frac{w_i^t}{\sum w_j^t}$.

2. Call the *weak learner*, with P_t and get hypothesis h_t.

3. Compute the error of h_t on the sample : $\epsilon_t = \sum_{i=1}^m P^t(i)|h_t(x_i) - c(x_i)|$.

4. Set $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$. the ratio between the error and the accuracy.

5. Calculate a new vector of weights $w_i^{t+1} = w_i^t \beta_t^{1-|h_t(x_i)-c(x_i)|}$.

$$h^A(x) = \begin{cases} 1 & \sum_{t=1}^T (log\frac{1}{\beta_t})h_t(x_i) \geq \frac{1}{2}\sum log(\frac{1}{\beta_t}) \\ 0 & else \end{cases} \tag{4.2}$$

The intuition behind this algorithm is to increase the probability of samples on which there were a misclassification.
Next we discuss the performance of the algorithm.

Theorem 4.5 *Suppose that in each step t a weak learner generated hypothesis h_t. The error of each hypothesis is denoted by ϵ_t. Given T errors $\epsilon_1 \ldots \epsilon_T$, the error ϵ of h^A is bounded by:*

$$\epsilon \leq 2^T \prod_{t=1}^T \sqrt{\epsilon_t(1-\epsilon_t)}$$

Proof: Denote $l_i^t = 1 - |h_t(x_i) - c(x_i)|$. Since the expression $|h_t(x_i) - c(x_i)|$ represents the error of a single sample, by multiplying the vector l_t by the probability vector p_t we get the average error ϵ_t.

$$1 - \epsilon_t = 1 - \sum_{i=1}^{m} p_i^t |h_t(x_i) - c(x_i)| = \sum_{i=1}^{m} p_i^t \{1 - |h_t(x_i) - c(x_i)|\} = P^t l^t$$

When we analyzed *Hedge*, we showed that

$$\sum_{i=1}^{m} w_i^{t+1} \le (\sum w_i^t)(1 - (1 - \beta_t) l^t P^t)$$

and thus

$$\sum_{i=1}^{m} w_i^T \le \prod_{t=1}^{T} (1 - (1 - \beta_t)(1 - \epsilon_t)).$$

The hypothesis h_A will make an error, only if

$$\prod_{t=1}^{T} \beta_t^{-|h_t(x_i) - c(x_i)|} \ge (\prod_{t=1}^{T} \beta_t)^{-\frac{1}{2}}$$

(because $c(x_i) \in \{0, 1\}$). The final weight of the point i, is :

$$w_i^{T+1} = D(x_i) \prod_{t=1}^{T} \beta_t^{1 - |h_t(x_i) - c(x_i)|} \ge (\prod_{t=1}^{T} \beta_t)^{\frac{1}{2}} D(x_i)$$

When we sum over all points we have:

$$\sum_{i=1}^{m} w_i^{T+1} \ge \sum_{i \in S} D(x_i) (\prod_{t=1}^{T} \beta_t)^{\frac{1}{2}} = \epsilon \sqrt{\prod_{t=1}^{T} \beta_t},$$

We already bounded the sum of the weights, and thus:

$$\epsilon \le \prod_{t=1}^{T} \frac{1 - (1 - \epsilon_t)(1 - \beta_t)}{\sqrt{\beta_t}}.$$

Since every element in the product is positive, we can minimize each element. By setting the derivative of each element to zero we can find the optimal β_t.

$$\frac{\partial}{\partial \beta_t} \frac{1 - (1 - \epsilon_t)(1 - \beta_t)}{\sqrt{\beta_t}} = \frac{\epsilon_t}{1 - \epsilon_t}.$$

We get:

$$1 - (1 - \epsilon_t)(1 - \beta_t) = 2\epsilon_t$$

This implies that our error bound is:

$$\epsilon \leq \prod 2\sqrt{\epsilon_t(1 - \epsilon_t)}.$$

■

Assuming that for all t's we have $\epsilon_t = \frac{1}{2} - \gamma$, we get the bound:

$$\epsilon \leq (1 - 4\gamma^2)^{\frac{T}{2}} \leq e^{-2\gamma^2 T}$$

Note however, that AdaBoost takes into account the accuracy of different hypothesis, and therefore in practise it might set much better bounds.