CS 596: Theoretical Machine Learning

Lecturer: Pranjal Awasthi — Lecture 5#
Scribe: Qiaoying Huang — Oct 04, 2016

The main topic of this lecture is the Online Learning Model. As opposed to the PAC model, here the algorithm gets to see examples in a sequence. At each step, the algorithm predicts the label for the given example and gets feedback on its performance. There are several reasons to study this model

- No need to assume that the test distribution the same as train distribution.
- Computational issues – training set might be too big to fit in memory. Want to process it in an online fashion.
- Intriguing question as to whether a probability free model for learning can exist.

Mistake Bound (MB) Model As in the PAC model, we have an instance space X, function class as H, and target function as $h^* \in H$. In the MB model, learning occurs rounds. In each round t:
1. The learner gets an unlabeled example $x_t \in X$.
2. The learner predicts its label $y_t \in \{0, 1\}$.
3. The learner is told the correct label $y = h^*(x_t)$.

The goal is to minimize the number of mistakes. For an online learning algorithm A, we denote by $MB_A(H)$ the maximal number of mistakes A might make on a sequence of examples which is labeled by some $h^* \in H$. Define $MB(H)$ as the best mistake bound achievable by a deterministic algorithm for learning H. Let's look at a simple example.

Theorem 1. *Let H be the class of monotone disjunctions over $\{0,1\}^n$, i.e, functions of the form $\prod_{i \in S} z_i$, where $S \subseteq [n]$ and $z_i \in \{0, 1\}$. Then $MB(H) \leq n$.*

Proof. The algorithm start with hypothesis $h_1 = z_1 \vee z_2 \ldots \vee z_n$.

- At time t, predict $y_t = h_t(x_t)$.
- On a mistake, remove any variable from h_t that takes value 1 on x_t.

It is easy to see that the algorithm never makes a mistake on an example with true label 1, since the variables in the true function are always a subset of the current hypothesis. Furthermore, every mistake removes at least 1 variable. Hence the total number of mistakes is bounded by n. □

We've seen an algorithm that makes at most n mistakes. In fact, no deterministic algorithm can guarantee a mistake bound less than n.

Theorem 2. *For the class of monotone disjunctions, we have $MB(H) \geq n$.*

Proof. Imagine seeing the following sequence of examples:

$$\begin{gathered} e_1 = (1, 0, 0, \cdots, 0) \\ e_2 = (0, 1, 0, \cdots, 0) \\ \vdots \\ e_n = (0, 0, 1, \cdots, 0) \end{gathered}$$

Any labeling of the above examples is consistent with some disjunction. So regardless of what the algorithm predicts, there is always a consistent disjunction that disagrees with the algorithm on every single example. □

Next we present a generic algorithm to learn any finite function class in the mistake bound model.

Halving Algorithm:

- Start with $H_1 = H$.
- At time t, predict the label of the majority vote in H_t.
- On a mistake, update H_t to H_{t+1} by deleting any function that was incorrect on x_t.

Theorem 3. *For any finite H, the Halving algorithm has a mistake bound $MB_A(H) \leq \log(|H|)$.*

Proof. To analyze the Halving Algorithm, let W_t = the number of surviving function in H after t rounds. Initially, $W_0 = |H|$.

$$\begin{gathered}\text{After 1 mistake, } W_1 \leq \tfrac{1}{2}|H| \\ \text{After 2 mistakes, } W_2 \leq \tfrac{1}{4}|H| \\ \vdots \\ \text{After } M \text{ mistake, } W_M \leq 2^{-M}|H|\end{gathered}$$

We know that one expert is perfect and will never be thrown out, so $W_M \geq 1$. Therefore, we have $1 \leq W_M \leq 2^{-M}|H|$ which implies the learner makes at most $\log(|H|)$ mistakes. □

Theorem 4 (Lower Bound)**.** *For any H, $MB(H) \geq d, d = VCdim(H)$.*

Proof If H has a VC dimension of d, this means that $\exists x_1, x_2, \cdots, x_d$ such that all 2^d labelings of this set can be realized by functions in H. In the online learning model, an adversary could force any learning algorithm A to make at least d mistakes by giving this shattered set as follows:

-Give A example x_1.
-A makes prediction $\hat{y}$.
-Choose to label x_1 with the opposite label $y \neq \hat{y}$.
-Do the same for the other $d-1$ examples in the shattered set.

The labeling that the adversary commits to will always be realizable.

Corollary 1. *For any finite H, $VCdim(H) \leq \log(|H|)$.*

Next, we provide a formal argument that the online learning model is harder than the PAC model. We will show how to convert an online learning model for a functions class into a PAC learning algorithm for the same class.

Theorem 5 (Online learning implies PAC learning)**.** *For any H if there exists an algorithm A with mistake bound M, then there exists an algorithm A', that PAC learns H provided $m \geq \frac{M}{\epsilon}\log(\frac{M}{\delta})$.*

Proof. Dram m i.i.d. examples and run A on them in an online fashion until it produces a hypothesis that survives the next $\frac{1}{\epsilon}\log(\frac{M}{\delta})$ examples, i..e. makes no mistake on them. The probability that a hypothesis with error rate$>\epsilon$ survives $\frac{1}{\epsilon}\log(\frac{M}{\delta})$ examples is at most $\frac{\delta}{M}$. There are at most M hypotheses generated by the algorithm[1], so, by union bound, the probability that the final hypothesis output by the above procedure has error rate$>\epsilon$ is at most δ. Also, since the mistake bound of A is M, if m is large enough then it must produce a hypothesis that survives $\frac{1}{\epsilon}\log(\frac{M}{\delta})$ examples. The sample complexity is $m \geq \frac{M}{\epsilon} log(\frac{M}{\delta})$ □

So far, we have assumed that the target function h^* is in H. As in PAC learning, we will now see what guarantees one can provide if $h^* \notin H$. We will first see how well the Halving algorithm performs in this case. In order to run the algorithm in this case, whenever we remove all the functions from consideration, the algorithm simply resets from the beginning. We then have the following bound

Theorem 6. *Given H and a sequence of examples S, let OPT=the number of mistakes of best function in H on S. Then, the Halving algorithm makes at most* $\log(|H|)(OPT+1)$.

Proof. As before, whenever the algorithm makes a mistake, we eliminate half of the hypotheses, and so the algorithm can make at most $\log(|H|)$ mistakes between any two resets. But if we reset, it is because since the last reset, every hypothesis has made a mistake: in particular, between any two resets, the best hypothesis has made at least 1 mistake. This gives the claimed bound □

We now give a much improved bound via an algorithm known as *Multiplicative Weights*. The algorithm maintains a weight for each function in the class and updates weights after every request.

Multiplicative Weights (MW)

- Initialize $w_h^1 = 1$, for all $h \in H$.
- At time t and input x_t, let $W^+ = \sum_{h\in H, h(x_t)=1} w_h^t$ and $W^- = \sum_{h\in H, h(x_t)=0} w_h^t$. Predict 1 if $W^+ \geq W^-$ and 0 otherwise.
- For every function h that made a mistake at time t, $w_h^{t+1} = \frac{1}{2}w_h^t$.

Theorem 7. *Given H and a sequence of examples S, let $m =$ mistakes of the best function in H on S. Then the number of mistakes M made by MW is at most* $2.4(m+\log(|H|))$.

Proof. Let W^t be the total weight of the functions in H at time t. We will track how the total weight changes. Initially: $W^1 = |H|$. Next we claim that on every mistake the total weight goes down by a factor of at least 3/4. This is because more than half of the weight must have been on incorrect functions and this weight goes down by half. Hence, $w_{t+1} \leq \frac{w_t}{2} + \frac{1}{2}\cdot\frac{w_t}{2} \leq \frac{3}{4}w_t$. So, after making M mistakes on the sequence S

$$W^{final} \leq |H| \cdot (\tfrac{3}{4})^M$$

Suppose the best function h_{best} makes m mistakes. Then we have that $w_{h_{best}}^{final} = \frac{1}{2^m}$

$\Rightarrow |H|(\frac{3}{4})^M \geq \frac{1}{2^m}$
$\Rightarrow (\frac{4}{3})^M \leq 2^m|H|$
$\Rightarrow M\cdot\log(\frac{4}{3}) \leq m + \log(|H|)$
$M \leq 2.4(m+\log(|H|))$ □

[1] No need to change the hypothesis if no mistake is made.

1 Additional Readings

Survey by Avrim Blum: `http://www.cs.cmu.edu/~avrim/Papers/survey.pdf`.