CS 395T Computational Learning Theory

Lecture 4: September 19, 2005 — Lecturer: Adam Klivans

Scribe: Anup Rao

4.1 Overview

Today we will first discuss how to do boosting assuming that we have an even better weak learner than usual. Specifically we will assume that the weak learner learns with a good advantage on both the positive and negative examples.

Then we will move on to start talking about learning concepts in the presence of *noise*.

4.2 Boosting via Branching Programs

4.2.1 The Assumption

Let c be concept that we are trying to learn. The assumption that we will work with is that we have a weak learner WL with the property that for every distribution D on inputs, the weak learner reads a few examples according to D and then outputs a hypothesis h s.t.

- $\Pr_{D^+}[h(x) = c(x)] \geq 1/2 + \gamma$
- $\Pr_{D^-}[h(x) = c(x)] \geq 1/2 + \gamma$

where γ is the advantage of the learner and D^+ (resp. D^-) represents the distribution D conditioned on the event that $h(x) = +$ (resp. $h(x) = -$) for x drawn from D. Formally D^+ is the random variable $D|h(D) = +$ and D^- is the random variable $D|h(D) = -$.

4.2.2 The Branching Program

First let us informally define what a branching program is. A branching program is an acyclic directed graph with a single start node (a.k.a vertex in the graph). Each node is labelled by a function which decides where to funnel the input. The value of the program on a particular input is computed by first placing the input at the start node. Based on the value of the function for the start node, the input is then moved to one of the neighbors of the start node. In this way in each step the input is moved from a node to a neighbor until it reaches a node which has no out edges. All such nodes are labelled by an output value. The value of the program for the input is simply the label of the final node that the input reaches.

We will build the following branching program:

For a parameter T (the depth of the branching program), the number of nodes will be $O(T)$. Each node will be indexed by $(i, j) \in \{(x, y) \in [T] \times [T] | x \leq y\}$. We will connect the nodes in the following way. For every node $V_{i,j}$ in the program, we connect it to $V_{i+1,j}$ (with a $-$ edge) and to $V_{i+1,j+1}$ (with a $+$ edge).

For every node $V_{i,j}$, we will assign a hypothesis $h_{i,j}$ and a distribution $D_{i,j}$. Given the weak learner WL as in the assumption, we will describe a process to compute $h_{i,j}$. Our final output hypothesis is then the branching program with the hypothesis $h_{i,j}$.

First let us describe how to use the $h_{i,j}$'s to label a particular input. Then we will discuss how to use the weak learner to come up with the $h_{i,j}$'s. Finally we shall prove that our learner actually does come up with a good hypothesis.

Using the $h_{i,j}$'s as a hypothesis

Recall that our branching program had a depth parameter T. The way we will label a particular input using our hypothesis is as follows. We first place the input x at the node $V_{1,1}$. If $h_{1,1}(x) = +$, we take the $+$ edge out of $V_{1,1}$ and move to $V_{2,2}$. In this way we traverse the branching program using x. In each step we are at a particular node $V_{i,j}$ and we evaluate $h_{i,j}(x)$ to figure out which of the two child nodes we should move to next. After T steps, we will be at a leaf of the branching program, i.e. some vertex $V_{T,j}$ for some j. At this point if $j \in [T/2]$, we output $-$, else we output $+$. The idea is that if most of the hypotheses that x encountered along the way labelled it $+$, the correct label is probably $+$. Otherwise the correct label is probably $-$. Essentially we are taking a majority of the hypotheses that get to see x. It is not immediately clear that this does anything useful. Note that each input x is judged by a different set of hypotheses $h_{i,j}$.

Using WL to come up with the $h_{i,j}$'s

Here we decribe how to computer the hypotheses that label the nodes of the branching program. The idea is simple. To compute the hypothesis $h_{1,1}$, we take several input examples according to D and feed it to the weak learner WL. WL computes some hypothesis that satisfies our assumption. Call this hypothesis $h_{1,1}$.

Next to compute $h_{2,1}$, we sample points according to D and evaluate $h_{1,1}$ on them. If $h_{1,1}$ evaluates to $-$, we feed the example to WL. In this way we effectively change the input distribution that WL sees to $D|h_{1,1}(D) = -$. Again for this distribution WL outputs some hypothesis which we call $h_{2,1}$. In this way we can compute the hypotheses for the second level of the tree. Repeating this process, we computer a hypothesis for every node of the branching program. So for every node, we first compute the hypotheses for all ancestors. Then we sample from our original distribution conditioned on the event that our input reaches the current node. We use WL to output a hypothesis that has a good advantage for this distribution and use this hypothesis to label the node. If some node is such that it takes too many samples from D to get a sample that reaches the node, we can ignore the node, since this means that our final hypothesis will generalize well regardless of how we label this node.

Proof that this works

Claim 1 *If we choose T to be $\log(1/\epsilon)/\gamma^2$, then the above branching program has accuracy $1-\epsilon$ with high probability.*

To prove the claim, for each node in the branching program, associate the distribution $D_{i,j}$ which is the distribution of D conditioned on the event that the input reaches the node $V_{i,j}$. To prove this we will need the concept of a martingale. The following definitions are taken from Motwani and Raghavan's book *Randomized Algorithms*.

First let us define the conditional random variable $E[X|Y]$.

Definition 1 (Conditional expectation) *The random variable $E[X|Y]$ is defined to be the random variable $f(Y)$ such that $f(y) = E[X|Y=y]$.*

Next we define a martingale.

Definition 2 (Martingale) *A sequence of random variables $X_0, X_1, \ldots$ is said to be a* martingale sequence *if for all $i > 0$,*

$$E[X_i|X_0, \ldots, X_{i-1}] = X_{i-1}$$

We will need the following tail inequality for martingales:

Theorem 1 (Azuma's Inequality) *Let $Y_0, Y_1, \ldots$ be a martingale sequence such that for each k,*

$$|X_k - X_{k-1}| \leq c_k$$

where c_k may depend on k. Then, for all $t \geq 0$ and any $\lambda > 0$,

$$\Pr[|X_t - X_0| \geq \lambda] \leq 2 \exp\left(- \frac{\lambda^2}{2 \sum_{k=1}^{t} c_k^2} \right)$$

Now we can prove that our hypothesis is accurate with high probability.

Proof: [Proof of Claim] Let Y be the random variable s.t. $V_{T,Y}$ is the final vertex that the input chosen according to D reaches. We need to show that when the input comes from the positive examples of D, $Y \geq T/2$ with high probability and that when the input comes from the negative examples of D, $Y < T/2$ with high probability.

To do this we will define a martingale sequence and use Azuma's inequality. Let $Y_0 = E[Y]$. For $i = 1, 2, \ldots, T$, let $Y_i = E[Y|X_1, \ldots, X_i]$. Thus $Y_T = Y$. Then it is easy to see that for every i, $E[Y_i|Y_0, \ldots, Y_{i-1}] = Y_{i-1}$.

Also $|Y_i - Y_{i_1}| \leq 1$. Thus by Azuma's inequality,

$$\begin{aligned}
&\Pr[|Y - E[Y]| \geq \gamma T] \\
&= \Pr[|Y_t - Y_0| \geq \gamma T] \\
&\leq 2\exp\left(-\frac{\gamma T^2}{2T}\right) \\
&= 2\exp\left(-\frac{\gamma T}{2}\right)
\end{aligned}$$

Thus Y deviates from its expectation with really small probability. However note that for positive examples, $E[Y] \geq T/2 + \gamma T$ and for negative examples $E[Y] \leq T/2 - \gamma T$. To see this, for positive examples, let X_i denote the random variable that denotes the event that the example was classified positive, i.e. X_i is 1 if the input picks the positive edge at the i'th level of the tree and 0 if it picks the negative edge. Then for positive inputs $E[X_i] \geq 1/2 + \gamma$. Thus by linearity of expectations, $E[Y] = E[\sum_i X_i] \geq T/2 + \gamma T$. We get a similar equation for negative examples.

Thus by our arguments above, the probability that our hypothesis misclassifies examples is small. ■

4.3 Learning with Noise

So far all the learning models we have looked at involved getting examples of some form that were correctly labelled. Now we will weaken the assumption that the labelling is correct.

In the standard PAC model we are given an example oracle which takes as input a distribution D and a concept c and produces examples according to D with labels according to c.

Now we consider the following model. In addition to the above inputs, the example oracle gets a parameter $1/2 > \eta \geq 0$. Now every time the example oracle is queried it gives a point x drawn according to D. However, the label that accompanies x is now correct (i.e. $c(x)$) with probability $1 - \eta$ and is incorrect with probability η. The goal as before is to output a hypothesis which is accurate with respect to the distribution D.

Let us see if we can learn some of our familiar concept classes in this more general model.

Conjunctions Recall our old learning algorithm: we maintain a table of variables (and their negations) and eliminate all variables which are set to false when the label is true. This no longer works, since we might eliminate a variable that we shouldn't have eliminated.

Axis Parallel Rectangles Again our old algorithm of finding the smallest rectangle which contains all the positive examples doesn't work.

We will show that we can still learn the above concepts in this model.

Conjunctions For every variable z, let $P_0(z)$ denote the probability that z is set to 0 with respect to D. Let's call z *significant* if $P_0(z) \geq \epsilon/8n$.

For each literal z, let $P_{01}(z)$ be the probability that z is set to 0 but the conjunction is true. If $P_{01}(z) \geq \epsilon/8n$ we will call z *harmful*.

Claim 2 *If h is a hypothesis which contains only significant unharmful variables, h is $1-\epsilon/2$ accurate.*

Proof: First note that h can disagree with the true concept in two ways. Either $c(x) = 1$ and $h(x) = 0$, or $c(x) = 0$ and $h(x) = 1$.

$\Pr_{x \leftarrow_{\mathrm{R}} D}[c(x) = 0 \text{ and } h(x) = 1] \leq (2n)(\epsilon/8n) \leq \epsilon/4$, by the union bound over all the $2n$ variables.

Similarly $\Pr_{x \leftarrow_{\mathrm{R}} D}[c(x) = 1 \text{ and } h(x) = 0] \leq (2n)(\epsilon/8n) \leq \epsilon/4$, by the union bound over all the $2n$ possible harmful variables.

■

Thus all we need to do to learn conjunctions with the noise is to determine which variables and insignificant or harmful and eliminate them. Note that for every variable z, we can estimate $P_0(z)$ and $P_{01}(z)$ just by taking several samples and counting the number of times that each event occurs.

Axis Parallel Rectangles This concept class can also be learned in the noise model using ideas similar as above. Filling in the details is left as an exercise to the reader.

The above kinds of algorithms for learning are called *statistical query algorithms*. In the next few lectures we will define and discuss this class of learning algorithms.