15-850(C) Machine Learning Theory

Lecturer: Avrim Blum Lecture 14: November 2, 1995

This topic is in the book, so only rough notes here....

Today's topic: Weak and Strong Learning

1 Intro

Say algorithm A weak-learns class C if for all $c \in C$, all distributions D, THERE EXISTS $\epsilon, \delta > 1/poly(n, size(c))$ such that A achieves error $1/2 - \epsilon$ with probability at least δ (and A uses time and number of examples $poly(n, size(c))$). In other words, with some probability you get noticeable correlation. For example, when we discussed lower bounds for SQ learning, we said that if there were too many orthogonal functions under some distribution, you can't even get this kind weak bias for that distribution.

Today we'll show the ability to weak learn a class C under arbitrary distributions implies "strong learning": i.e., our usual definition of learning.

We already saw how to boost δ: we run the algorithm a bunch of times and test hypotheses. The real problem is boosting ϵ.

In the distribution specific case, you can't do it. Imagine the concept "$x_1 \wedge$(hard)", and learning over the uniform distribution. We can easily get an error rate of 25% (when x_1 is off we say negative, and otherwise we guess) but it will be hard to do better.

To show that weak learning implies strong learning, we use the fact that the learner must get weak learning over ALL distributions.

We will be able to view our proof as a "meta" algorithm: we're given a black box learning algorithm that achieves some accuracy, and we want to either automatically boost up our accuracy or else manage to find a disribution that's hard for it (which might help us in modifying our algorithm or understanding our data better).

2 An easy case

Suppose our weak-learning algorithm only makes one-sided error. For instance, it gets all of the positives right and at least $1/poly$ of the negatives right. (I.e., when the hypothesis says "negative" then it's always right, but when it says positive it might or might not be right). Then, boosting is easy:

We first find h_1 that gets at least $1/poly$ of the negatives right over distribution D. We then define D_2 to be D restricted to examples x such that $h_1(x) = 1$ and learn over that, etc.

Our final hypothesis is the AND of all the $h_i(x)$'s.

For boosting hypotheses that make 2-sided error, though, it's a little more tricky.

3 The basic idea

There are several methods for doing this. We'll talk about one, but more efficient methods have been developed. Some of the details of the analysis are messy. So, to focus on key ideas, we will run through a case with specific numbers. Let's suppose we have an algorithm A that over any distribution will get at least 70% accuracy with high probability. Say we want to boost it up a bit (which we can then apply recursively). This will demonstrate the main ideas.

We start by running A on initial distribution $D_1 = D$ and getting hypothesis h_1. Say its accuracy is 70%. Now we want a new distribution.

One try: filter D to only look at examples where h_1 predicts incorrectly and try to learn over that. This DOESN'T work. Why?

Instead: we filter to create a distribution where h_1 behaves like random guessing. Specifically, let S_C be the set of examples on which h_1 predicts correctly and S_I be the set on which h_1 predicts incorrectly. What we want is a distribution D_2 where S_C and S_I both have equal weight, but besides that is just like D_1 in that $\Pr_{D_2}(x|x \in S_C) = \Pr_{D_1}(x|x \in S_C)$ and the same for S_I. For example, we can construct D_2 by flipping a coin and if heads we wait until we see an example from D_1 where h_1 is correct, and if tails we wait until we see one where h_1 is incorrect. Or we can draw examples from D_1 and place them into two queues, one for S_C and one for S_I, and then to get an example from D_2 we randomly choose a queue to select from.

Let's write $D_2[x]$ as a function of $D_1[x]$ (this is the probability associated with x, or the density function if we're in a continuous space). To create D_2 we reduced the probability on S_C from 70% to 50%, so if $x \in S_C$ then $D_2[x] = (5/7)D_1[x]$. We increased the probability on S_I from 30% to 50% so if $x \in S_I$ then $D_2[x] = (5/3)D_1[x]$.

Now we get a hypothesis h_2 with accuracy 70% on D_2.

Finally, we filter D only allowing examples x where $h_1(x) \neq h_2(x)$. We then get hypothesis h_3 with accuracy 70% on this distribution D_3.

Resulting hypothesis: if h_1 and h_2 agree, then go with that, else go with h_3. I.e., majority vote.

4 Analysis

What's the accuracy of the new hypothesis?

Easiest way is to draw a picture. Divide space into 4 regions: R_1 in which $h_1 = c$ and $h_2 = c$,

R_2 in which $h_1 = c$ and $h_2 \neq c$, R_3 in which $h_1 \neq c$ and $h_2 \neq c$, and R_4 in which $h_1 \neq c$ and $h_2 = c$.

We predict correctly on R_1, incorrectly on R_3 and get 70% correct on $R_2 \cup R_4$. Let's say that $D_2[R_2] = \gamma$. By the fact that h_2 has 30% error on D_2 we know that $D_2[R_3] = 0.3 - \gamma$, and by definition of D_2 we know that $D_2[R_1 \cup R_2] = D[R_3 \cup R_4] = 1/2$, so

$$D[R_1] = \frac{7}{5}(1/2 - \gamma)$$

and

$$D[R_3] = \frac{3}{5}(3/10 - \gamma).$$

We can now work out our failure probability as:

$$\begin{aligned}
\Pr_D[\text{fail}] &= D[R_3] + (3/10)(1 - D[R_1] - D[R_3]) \\
&= (7/10)D[R_3] + (3/10)(1 - D[R_1]) \\
&= (7/10)\left[\frac{3}{5}(3/10 - \gamma)\right] + (3/10)\left[3/10 + (7/5)\gamma\right] \\
&= \frac{3}{10}((7/5)(3/10) - (7/5)\gamma + 3/10 + (7/5)\gamma) \\
&= (3/10)^2[3 - 3/5] \\
&\approx 0.216
\end{aligned}$$

More generally, if we work this through, we get:

$$\text{error} \leftarrow (\text{error})^2[3 - 2 \times \text{error}]$$

This is always a decrease assuming that our original error was strictly between 0 and 1/2.

5 Details

If our hypotheses h_i did better than 70% accuracy, then it's not hard to see that this method does even better. As we get accuracies close to 1, then it gets harder to simulate the distribution D_2, but this extra time is linear in $1/\epsilon$ so it's OK.

If our original error was less than 1/4, then our error roughly squares each time we use this procedure so we need $\log\log(1/\epsilon)$ levels of recursion to get desired error ϵ. Otherwise we need more like $\log(1/\epsilon)$ levels of recursion (meaning $poly(1/\epsilon)$ applications of our original algorithm).

Another thing to worry about is recursive filtering, but that is OK too.

6 An application

Given a concept class C and a distribution D, we can define the "statistical-query dimension" of C with respect to D as the largest d such that there exist d functions $f_1, f_2, \ldots, f_d \in C$ such that $\langle f_i, f_j \rangle_D \leq 1/d$ for all $i \neq j$.

For a given class C, if there exists a distribution D such that SQ-DIM(C, D) is larger than any polynomial in our desired parameters, then you can't learn C under D in the SQ model. But, suppose that for all D we have that SQ-DIM(C, D) is polynomial AND we have some algorithmic way of finding the f_i's. Then an algorithm is:

1. Find the f_i's.
2. Notice that the target must now have at least a $1/poly$ correlation with at least one of the f_i's, and so we just test them all out to find one. This gives us weak learning which then implies strong learning.