15-850(C) Machine Learning Theory

Lecturer: Avrim Blum Lecture 6: October 5, 1995

1 On-line learning as a 2-person game

If we ignore the issue of computational efficiency. we can think of learning a concept class C in the mistake-bound model as a 2-person game. In particular, for a deterministic algorithm, the game goes like this:

1. The adversary selects some example x, which splits C into two sets: C_0 is the set of concepts that label x negative, and C_1 is the set of concepts that label x positive.
2. The algorithm gets to decide to keep one of C_0 and C_1 and throw away the other (i.e., by predicting postive and making a mistake it throws out C_1, or by predicting negative and making a mistake it throws out C_0).
3. We replace C by whichever of C_0 or C_1 was kept. If C still has more than one concept in it, we go back to Step (1).

The value of this game (to the adversary) is the number of rounds the game is played.

Looking at the problem this way, we can see a couple of things. First, there is a well-defined notion of optimal strategies for each player. That is, there is a well-defined number $OPT(C)$ that is the optimal mistake bound for concept class C. Second, what is this optimal strategy? Well, given an example x, we "just" calculate $OPT(C_0)$ and $OPT(C_1)$ (by applying this idea recursively),[1] and we predict on x based on whichever is larger. I.e., if $OPT(C_0) \leq OPT(C_1)$ then we predict positive. This is often called the *standard optimal algorithm*. Notice that this is *not* necessarily the same thing as the Halving Algorithm we discussed last time.

We can think of a fixed adversary strategy as defining something called a "mistake tree". Each node of this tree contains an example. The root contains the initial example given to the learner. The "0" branch leads to the tree of examples that are given if the learner predicts negative, and the "1" branch leads to the tree of examples given if the learner predicts positive. For a given mistake tree, it's not hard to see that the best learner strategy is to pick the direction in which the subtree has least *rank*, and the number of mistakes made is exactly the rank of the tree. So, $OPT(C)$ is the maximum, over all possible mistake trees for C, of the rank of that mistake tree.

2 The perceptron algorithm

See slides.

[1] What is really being applied recursively is the formula: $OPT(C) = 1 + \max_x \min\{OPT(C_0), OPT(C_1)\}$.