

15-859(B) Machine Learning Theory

Homework # 5

Solutions

Exercises:

1. **[SQ model]** Give an algorithm to learn the class of *decision lists* in the SQ model (and argue correctness for your algorithm). Be clear about what specifically the queries χ are and the tolerances τ . Remember, you are not allowed to ask for conditional probabilities like $\Pr[A|B]$ but you can ask for $\Pr[A \wedge B]$.

So, combined with your results from Homework 1, this gives an algorithm for learning decision trees of size s in the SQ model with queries and τ^{-1} on the order of $n^{O(\log s)}$, matching our SQ-dimension lower bounds.

Solution: We can just use the standard algorithm, but we need to phrase it in the SQ language. Let r_1, r_2, \dots, r_{4n} be all $4n$ possible rules. (Assume we have a fake variable that is always on, just to make the number of rules a round number). We begin by asking, for each rule, “what is the probability a random example satisfies r_i but r_i gives the wrong label?” In other words, we ask the $4n$ queries: $\Pr[x_i = 1 \wedge c(x) = 1]$, $\Pr[x_i = 1 \wedge c(x) = 0]$, $\Pr[x_i = 0 \wedge c(x) = 1]$, and $\Pr[x_i = 0 \wedge c(x) = 0]$. We ask these queries with tolerance $\frac{\epsilon}{8n}$. We know that one of these questions (the one that asks about the top rule in the target function) has true answer of 0, so we are guaranteed that at least one query will give us an answer $\leq \frac{\epsilon}{8n}$. Let us call one such rule $r_{(1)}$. We put $r_{(1)}$ at the top of our list (incurring an error at most $\frac{\epsilon}{4n}$), and then ask, for each remaining rule r_i , “what is the probability a random example satisfies $(\neg r_{(1)}) \wedge r_i$ but r_i gives the wrong label?” with tolerance $\frac{\epsilon}{8n}$. Again we are guaranteed that for one of these queries, the correct answer is 0, so at least one rule $r_{(2)}$ will give us an answer at most $\frac{\epsilon}{8n}$ and we can put it as the second rule, incurring an additional error at most $\frac{\epsilon}{4n}$. More generally, our query is “what is the probability a random example satisfies $(\neg r_{(1)}) \wedge (\neg r_{(2)}) \wedge \dots \wedge (\neg r_{(k)}) \wedge r_i$ and yet r_i gives the wrong label. Since there are at most $4n$ rules, our overall total error is at most ϵ .

Problems:

In the rest of this homework you will show that the class of polynomial-size boolean formulas is equivalent to the class NC^1 . NC^1 is the class of $O(\log n)$ -depth $\{\text{AND}, \text{OR}, \text{NOT}\}$ circuits where each gate has at most 2 inputs. Boolean formulas are just the generalization of DNF in which we allow ANDs and ORs to appear in any order, e.g., $x_1 \vee (x_2 \wedge (\bar{x}_1 \vee x_3))$; you can think of a Boolean formula as a circuit that looks like a tree, except you are allowed to have multiple copies of each input (or equivalently, the inputs are allowed to have out-degree greater than 1. This is also exercise 6.2 (p.141) in the book.

2. **[circuits and formulas part 1]** Show that for any circuit of depth d of $\{\text{AND}, \text{OR}, \text{NOT}\}$ gates with fanin ≤ 2 , there is an equivalent boolean formula of size $O(2^d)$. Thus, NC^1 is contained in the class of polynomial-size boolean formulas.

Solution: Just to be specific, let's say the "size" of a formula is the number of nodes in the tree (so " $x_1 \vee x_2$ " has size 3, since we include 1 for the " \vee "). We'll construct a boolean formula of size at most $2^{d+1} - 1$.

Proof is by induction. Base case: a single computational node ($d = 1$) is a formula of size 3, so that's OK. General case: by induction, each subtree of the root can be replaced by a formula of size at most $2^d - 1$, so the total size of the resulting tree (including the root) is at most $1 + (2^d - 1) + (2^d - 1) = 2^{d+1} - 1$ as desired.

3. **[circuits and formulas part 2]** Show that for any boolean formula of size s , there exists an equivalent circuit of depth $O(\log s)$. Thus polynomial-size boolean formulas are contained in the class NC^1 .

Solution: In any tree of size s , there must exist a node whose removal splits the tree into pieces of size at most $s/2$. (To find this node, pick one arbitrarily to start with; if this node does not have the desired property, move to its neighbor in the direction of the largest subtree and repeat. Notice that each iteration reduces the size of the largest subtree by at least 1, and the other subtrees cannot grow to larger than $s/2$, so this process must halt.)

Given a boolean formula of size s , think of it as a tree T and find this vertex v . Let's say v is an OR node (the case of an AND node is analogous) with subtrees A and B . Let C_0 be the tree T with v replaced by a 0 (and A and B removed), and let C_1 be the tree T with v replaced by a 1. Then, T can be rewritten as

$$((A \vee B) \wedge C_1) \vee (\bar{A} \wedge \bar{B} \wedge C_0).$$

Thus, using 3 levels we have reduced the problem to converting trees of size at most $s/2$ to circuits (we can ignore negations since they can all be moved down to the leaves). Thus, continuing recursively on the subtrees, the total depth is at most $3 \log_2 s$.

Thus, this implies our hardness results for learning NC^1 (from class on Feb 27) carry over to general Boolean formulas (can't even weak-learn in poly-time over the uniform distribution, even with membership queries, under cryptographic assumptions). Note: Problem 2 should be pretty straightforward. Problem 3 is trickier.