TTIC 31250 An Introduction to the Theory of Machine Learning

Homework # 4 **Solutions**

Groundrules: Same as before.

Problems:

1. [**PAC-learning of small OR-functions**] Suppose the target function is a disjunction (OR-function) of r out of n boolean variables, where r is much less than n. For example, perhaps only $\sqrt{n}$ of the n variables are used in the target ($r = \sqrt{n}$). The list-and-cross-off algorithm for learning an OR function that we gave in class might produce a hypothesis of size $O(n)$, so its sample size for PAC-learning would be $\tilde{O}(n/\epsilon)$. Alternatively, we could try all $O(n^r)$ possible OR-functions of size $\leq r$ and pick one consistent with our training sample. Since $\log|H| = r\log n$, this would require only $\tilde{O}((r\log n)/\epsilon)$ samples but takes time exponential in r.

   Your job: Give a polynomial-time algorithm that guarantees to find an OR of at most $O(r\log m)$ variables that is consistent with the training data, where m is the number of training examples. So this is not quite finding the *smallest* OR-function, but it's close. Describe a variant of your algorithm (also running in polynomial time) that finds an OR of only $O(r\log(1/\epsilon))$ variables, with error at most $\epsilon/2$ on the training data. Note that by Occam + Chernoff bounds, the latter algorithm requires sample size at most $O(\frac{1}{\epsilon}((r\log 1/\epsilon)\log n + \log 1/\delta))$ to learn in the PAC model, so its sample complexity is just very slightly worse than the exponential-time algorithm that tries all $O(n^r)$ OR-functions of size $\leq r$.

   Hint: think about the greedy set-cover algorithm (look it up if you haven't seen it).

   Solution: first, remove any variable that is set to 1 by any negative example, since we know it cannot be in the target. Now, for the first question, run greedy set-cover: specifically, select the variable that is set to 1 by the most positive examples in the training data, remove those examples from the data set, and repeat. Since every positive example sets at least one of the r relevant variables to 1, there must always be a variable to select that covers (i.e., is set to 1 by) at least a $1/r$ fraction of the positive examples remaining. Therefore after t variables have been selected, the number of positive examples remaining is at most $\lfloor m(1-1/r)^t \rfloor$, which is 0 for $t = r\ln m$. For the second question, just use the same procedure, but halt when at most $\epsilon m/2$ positive examples are left. By the same reasoning, this will halt after at most $t = r\ln(2/\epsilon)$ steps.

2. [**Uniform distribution learning of DNF Formulas**] Give an algorithm to learn the class of *DNF formulas* having at most s terms over the *uniform distribution* on $\{0,1\}^n$, which has sample size polynomial in n and s, and running time $n^{O(log(s/\epsilon))}$. So, your algorithm matches the SQ-dimension lower bounds.

   Hint: Think of your algorithm for Problem 1.

Note: your solution requires that data come from the uniform distribution. The best algorithm known for learning polynomial-size DNF formulas over general distributions has running time roughly $2^{O(n^{1/3})}$ [Klivans-Servedio].

Solution: define variables $z_1, z_2, \ldots, z_t$ corresponding to all terms of size up to $\lg(2s/\epsilon)$. So, $t = O(n^{\lg(2s/\epsilon)})$. Because examples are from the uniform distribution, any given term of larger size has probability at most $\epsilon/(2s)$ of being satisfied, and thus the OR of all such terms is satisfied with probability at most $\epsilon/2$. This means there exists a disjunction of at most s variables z_i of error at most $\epsilon/2$ (all on positive examples misclassified as negative) and if we draw a sufficient size sample, with high probability this disjunction will have error at most $3\epsilon/4$ on the sample. Thus, running the greedy algorithm from Exercise 1 we can produce a disjunction of size $O(s \log 1/\epsilon)$ of training error at most ϵ. Overall, our sample size needed is linear in the size of the disjunction found and logarithmic in the total number t of variables z_i, giving us our desired bound.

3. **[SQ learning Decision Lists and Trees]** Give an algorithm to learn the class of *decision lists* in the SQ model (and argue correctness for your algorithm). Your algorithm should work for any distribution D (not just the uniform distribution). 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 $n^{O(\log s)}$ queries of tolerance $1/n^{O(\log s)}$, matching our SQ-dimension lower bounds.

   Note: this problem can be tricky. In particular, it is possible to create a distribution D over $\{0,1\}^n$ and a target decision list c with the following properties:

   (a) $\Pr_D[c(x) = 1] = 1/2$.

   (b) For all $1 \leq i \leq n$, either $\Pr_D[x_i = 1] \leq 2^{-n/2}$ or else $\Pr_D[c(x) = 1 | x_i = 1] = 1/2$.

   In particular, no variable is noticeably correlated with the target! (Any variable either is almost never 1 or else is completely uncorrelated with the target). So, an algorithm that tries to find x_i, y, b such that both (a) $\Pr[c(x) = y | x_i = b]$ is large and (b) "$x_i = b$" happens with noticeable probability is going to have trouble. In fact, this difficulty is one reason that there is no known analog of Problem 1 for decision lists. Instead, think about building on the algorithm from the very first lecture.

   Solution: We can just use the standard algorithm, but we need to phrase it in the SQ language. Let $r_1, r_2, \ldots, r_{4n}$ 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 proability 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 $(\neg r_{(1)}) \wedge (\neg r_{(2)}) \wedge \ldots \wedge (\neg r_{(k)}) \wedge r_i$ and yet r_i gives the wrong label. Since there are at most $4n$ rules, our overall total error is at most ϵ.