15-859(B) Machine Learning Theory

Homework # 5 **Due: April 5, 2006**

Groundrules: Same as before. You should work on the exercises by yourself but may work with others on the problems (just write down who you worked with). Also if you use material from outside sources, say where you got it.

In the first part of this assignment, we will derive an algorithm for learning decision trees of size s in time $n^{O(\log s)}$ in the PAC model (or with $n^{O(\log s)}$ mistakes and time-per-stage in the mistake-bound model). In fact, we will be able to do this with a statistical query algorithm. This matches the lower bound of $n^{\Omega(\log n)}$ for learning size-n decision trees with SQ algorithms that we proved in class.

Exercises:

1. A k-decision list is a decision list where each rule has as its precondition a conjunction of up to k literals. For instance, the function: "if $x_1\bar{x}_2$ then $+$, else if $\bar{x}_1 x_3$ then $-$, else if x_4 then $+$, else $-$" is a 2-decision list.

   (a) Describe briefly why the class of k-decision lists can be learned with mistake bound $O(n^{2k})$.

   (b) Describe how you can learn k-decision lists in the SQ model.

2. The *rank* of a decision tree is defined as follows. If the tree is a single leaf then the rank is 0. Otherwise, let r_L and r_R be the ranks of the left and right subtrees of the root, respectively. If $r_L = r_R$ then the rank of the tree is $r_L + 1$. Otherwise, the rank is the maximum of r_L and r_R. For example, the decision tree in Figure 2.2 in the book has rank 3.

   Prove that a decision tree with ℓ leaves has rank at most $\log_2(\ell)$.

Problems:

3. Show that the class of rank-k decision trees is a subclass of k-decision lists.

   Hint: Use induction on the depth of the tree. (Actually, there are several different ways of proving this.)

   Thus, we conclude that we can learn constant rank decision trees in polynomial time, and we can learn arbitrary decision trees of size s in time and number of examples $n^{O(\log s)}$. (So this is "almost" a PAC-learning algorithm for decision trees.)

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 {AND, OR, 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 the inputs are allowed to have out-degree greater than 1. This is also exercise 6.2 (p.141) in the book.

4. Show that for any circuit of depth d of $\{$AND, OR, 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.

5. 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.

Thus, this implies our hardness results for learning NC^1 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: question 4 should be pretty straightforward. Question 5 is trickier.