15-859(B) Machine Learning Theory

Homework # 1 **Due: January 26, 2009**

Groundrules:

- Homeworks will generally consist of *exercises*, easier problems designed to give you practice, and *problems*, that may be harder, and/or somewhat open-ended. You should do the exercises by yourself, but you may work with a friend on the harder problems if you want. (Working together doesn't mean "splitting up the problems" though.) If you work with a friend, then write down who you are working with.

- If you've seen a problem before (sometimes I'll give problems that are "famous"), then say that in your solution. It won't affect your score, I just want to know. Also, if you use any sources other than the textbook, write that down too. It's fine to look up a complicated sum or inequality or whatever, but don't look up an entire solution.

Exercises:

1. **Compression and learning.** In class we showed that so long as the number of examples m satisfies $m > \frac{1}{\epsilon}[s \ln 2 + \ln(1/\delta)]$, then with probability at least $1 - \delta$, all consistent hypotheses h with size$(h) < s$ have error at most ϵ. Equivalently, to learn well it suffices to have $s < \frac{1}{\ln 2}[\epsilon m - \ln(1/\delta)]$; i.e., to compress the data by roughly a factor of $1/\epsilon$.

   Suppose you have an algorithm that guarantees on any set of m examples consistent with some target function $f \in C$ to find a hypothesis of size at most size$(f)^3 m^{2/3}$. How large a value of m would be sufficient for this algorithm to achieve error $\leq \epsilon$ with probability $\geq 1 - \delta$. (You may look at the proof of Thm 2.1 in the book, but please don't just plug into the theorem. Also, don't worry about being tight with constant factors.)

2. **Expressivity of LTFs.** It is easy to see that conjunctions (like $x_1 \wedge \bar{x}_2 \wedge x_3$) and disjunctions (like $x_1 \vee \bar{x}_2 \vee x_3$) are special cases of decisions lists. Show that decisions lists are a special case of linear threshold functions. That is, any function that can be expressed as a decision list can also be expressed as a linear threshold function "$f(x) = +$ iff $w_1 x_1 + \ldots w_n x_n \geq w_0$".

3. **Decision tree rank.** 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.

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

Problems:

4. **Decision List mistake bound.** Give an algorithm that learns the class of decision lists in the mistake-bound model, with mistake bound $O(nL)$ where n is the number of variables and L is the length of the shortest decision list consistent with the data. The algorithm should run in polynomial time per example.

   Hint: think of using some kind of "lazy" version of decision lists as hypotheses.

   Note: there is a solution to this problem in the survey article handed out in the second lecture, but please do this yoursel{f,ves}.

5. **Expressivity of decision lists, contd.** Show that the class of rank-k decision trees is a subclass of k-decision lists. (There are several different ways of proving this.)

   Thus, we conclude that we can learn constant rank decision trees in polynomial time, and using Exercise 3 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.)

6. **Halving is not always optimal.** Describe a class C where the halving algorithm is not optimal: that is, where you would get a better worst-case mistake bound by *not* going with the majority vote of the available concepts.

   Note: it's OK if your class is a bit contrived.

   Hint: The key issue here is: what if on the current example x, 80% of the functions say "positive" and 20% say "negative", but the larger set of functions has a smaller mistake bound than the smaller set of functions? And how can a large set of functions have a small mistake bound anyway?