TTIC 31250 An Introduction to the Theory of Machine Learning

Homework # 1 **Solutions**

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 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 material handed out, 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. **Grading Signup.** Please sign up to grade one of the homeworks using this Google Form.

2. **Expressivity of LTFs.** Recall that a decision list is an if-then-else... function where each if-condition examines the value of a single variable, e.g., "if $x_4 = 0$ then $+$ else if $x_7 = 1$ then $-$ else ...". 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 $w_1x_1 + \ldots w_nx_n \geq w_0$".

   Solution: Let's first suppose there are no negated variables in the decision list. So, the decision list looks like "if x_{i_1} then y_1, else if x_{i_2} then y_2, else ...", where each $y_j \in \{+, -\}$. Let L be the length of the list (including the final "else"). In that case, we can use the linear threshold function:

   $$w_1x_1 + w_2x_2 + \ldots + w_nx_n + w_{n+1} > 0,$$

   where $w_{i_j} = 2^{L-j}$ if $y_j = +$, and $w_{i_j} = -2^{L-j}$ if $y_j = -$, and $w_i = 0$ if x_i was not in the list at all. Finally, $w_{n+1} = 1$ or -1 depending on whether the last rule is "else $+$" or "else $-$" respectively. Since the weights are dropping by factors of 2, the first variable set to 1 in the list will override all the ones after it. In particular, if the jth rule in the list is the first rule satisfied by a given example, then because $|w_{i_j}| > |w_{i_{j+1}}| + |w_{i_{j+2}}| + \ldots$, if w_{i_j} is positive then the LTF will be satisfied and if w_{i_j} is negative then it won't be satisfied, regardless of the values of the variables that come after.

If there are negated variables in the decision list, then we just want to view $\bar{x}_{i_j}$ as $1 - x_{i_j}$, in the LTF above. For example, if $n = 3$, the decision list "if x_1 then $+$, else if $\bar{x}_2$ then $-$, else $+$" would become $4x_1 - 2(1 - x_2) + 1 > 0$.

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 $\log_2(\ell)$.

   Solution: Proof by induction on the depth of the tree. Base case: it's true for a single leaf by definition of "rank". General case: say the given tree of depth d has rank r and its left and right subtrees have rank r_L and r_R respectively. Since each subtree has depth at most $d - 1$, by induction, the number of leaves is at least $2^{r_L} + 2^{r_R}$. This sum is at least 2^r by definition of "rank". So, by induction, a tree of rank r has at least 2^r leaves, which means that a tree with ℓ leaves has rank at most $\log_2(\ell)$.

   Note that it's a bit trickier to solve this problem by induction on rank, since the left and right subtree of a given tree might not both have strictly smaller rank than the overall tree.

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. Briefly explain how your algorithm can be extended to learn the class of k-decision lists with mistake bound $O(n^k L)$ (a k-decision list is a decision list in which each test is over a conjunction of k variables rather than just over a single variable).

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

   Solution: There are a few different ways to solve this problem. Here is one.

   Take all $4n + 2$ possible rules ($4n$ rules of the form "if $x_i = a_i$ then b_i" and two rules of the form "else b_i") and put them into a set S_1. Initialize sets $S_2, S_3, ..., S_{n+1}$ to be empty. Define hypothesis $h(x)$ as: "if any rule in S_1 fires, use it for prediction (if several fire, choose one that fires arbitrarily); else if any rule in S_2 fires, use it for prediction (again, if several in S_2 fire, choose one of them arbitrarily); else if any rule in S_3 fires,...". If $h(x)$ is incorrect, then take the rule that was used for prediction and move it into the next set: that is, if it was in set S_i then move it to set S_{i+1}.

   Let $f = (r_1, r_2, r_3, ..., r_L)$ denote the true target decision list, where r_i is the ith rule in the list . Notice that r_1 will never be moved down from S_1 because whenever it fires, it is correct (by definition of a decision list). Now, assume inductively that r_j is never moved below S_j for $j = 1, 2, ..., i$. Then r_{i+1} is never moved below S_{i+1} because if r_{i+1} ever reaches S_{i+1} and is used for prediction, it must be that *none* of $r_1, ..., r_i$ fired. So (by definition of a decision list) r_{i+1} fires correctly.

Since every mistake moves some rule down by one level, and none of the rules in f are ever moved below level L, this means that no rule will ever be moved past level $L + 1$. So, the total number of mistakes is at most $(4n + 2)L = O(nL)$.

To extend the result to k-decision lists, we just do the same as above but with $O(n^k)$ rules r_i instead of $O(n)$ rules.

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.)

   Solution: Proof by induction on depth. Base case (a single leaf) is true by definition. General case: say the tree has rank k and its root contains variable x_i. We know one of the subtrees has rank at most $k - 1$ (without loss of generality, say this is the left subtree) and the other has rank at most k. Inductively, say that L is a $(k - 1)$-decision list equivalent to the left subtree and L' is a k-decision list equivalent to the right subtree. To create a k-decision list for the entire tree, begin with the list L but with x_i appended onto each rule, followed by the list L'. Notice that we do not need to append $\bar{x}_i$ onto the rules of L' because every example with $x_i = 0$ exits through one of the rules of L.

   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.)