

15-859(B) Machine Learning Theory

Homework # 1

Solutions

Exercises:

1. **Expressivity of decision lists.** Show that conjunctions and disjunctions are both special cases of decision lists. That is, any function that can be expressed as a conjunction (or disjunction) can also be expressed as a decision list.

Solution: To express the disjunction $\ell_1 \vee \ell_2 \vee \dots \vee \ell_k$ (where each ℓ_i is a literal), you simply write the decision list: if ℓ_1 then +, else if ℓ_2 then +, ..., else if ℓ_k then +, else -.

To express the conjunction $\ell_1 \wedge \dots \wedge \ell_k$, you use the decision list: if $\bar{\ell}_1$ then -, else if $\bar{\ell}_2$ then -, ..., else if $\bar{\ell}_k$ then -, else +.

2. **Expressivity of LTFs.** Show that decision 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 $w_1x_1 + \dots + w_nx_n \geq w_0$ ”.

Solution: First of all, let's assume 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. In that case, we can use the linear threshold function:

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

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. The point is that since the weights are dropping by factors of 2, the first variable on in the list will override all the ones below it.

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 has rank r and its left and right subtrees have rank r_L and r_R respectively. By induction, the number of leaves is at least $2^{r_L} + 2^{r_R}$, which is at least 2^r by definition of “rank”.

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.

Solution: See page 10 of the handout “On-Line Algorithms in Machine Learning” for one solution to this problem.

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.)

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.

Solution: The main issue here is that it is possible for the *size* of a class C to not be a good indicator of how hard it is to learn. For example, let C_1 be the set of concepts over $\{0, 1\}^n$ that each have just one positive example. Then $|C_1| = 2^n$ but we can learn C_1 with a mistake-bound of 1 (just predict negative until you make your first mistake). To answer the question, we just union C_1 with a smaller class C_2 whose size *is* a good indicator of its difficulty to learn, and that disagree with most of C_1 on our initial example. For instance, let C_2 be the set of 4 OR-functions over the set $\{x_1, x_2, x_3\}$ that contain x_1 . If the initial example is $1000 \dots 0$, then the halving algorithm on $C = C_1 \cup C_2$ will predict negative (and have a worst-case mistake-bound of 3), whereas the optimal algorithm predicts positive and has a worst-case mistake-bound of 2.