TTIC 31250 : Introduction to Theory of Machine Learning

Spring 2020

•

Homework 1

•

Solutions by

Jinhong Du

dujinhong@uchicago.edu

12243476

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 $x_1 \wedge \overline{x}_2 \wedge x_3$) and disjunctions (like $x_1 \vee \overline{x}_2 \vee x_3$) 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 $w_1 x_1 + \ldots + w_n x_n \geq w_0$".

   Proof. Without loss of generality, we can consider the following decision list,

   $$\begin{aligned}
   &\texttt{if } x_n = 1 \texttt{ then } b_n \\
   &\texttt{else if } x_{n-1} = 1 \texttt{ then } b_{n-1} \\
   &\qquad\vdots \\
   &\texttt{else if } x_2 = 1 \texttt{ then } b_2 \\
   &\texttt{else if } x_1 = 1 \texttt{ then } b_1 \\
   &\texttt{else } -b_1
   \end{aligned}$$

   where $b_i \in \{+, -\}$. Otherwise, we can rearange the features and then transform some features by $x_i = 1 - x_i$, so that the decision list has the above form. We prove by induction.
   (1) When $n = 1$, the decision list is

   $$\texttt{if } x_1 = 1 \texttt{ then } + \texttt{ else } -$$

   then the linear threshold function is "$f_1(x_1) = +$ if and only if $x_1 \geq 1$".
   (2) Suppose that for $k \in \{1, \ldots, n-1\}$, the linear threshold function is $f_k(x_1, \ldots, x_k) = +$ if and only if

   $$\sum_{i=1}^{k} 2^{i-1} x_i \mathbb{1}_{\{b_i = +\}} - 2^{i-1}(1 - x_i)\mathbb{1}_{\{b_i = -\}} \geq 1,$$

   where $b_1 = +$. Then for $k+1$, the sub decision list is

   $$\begin{aligned}
   &\texttt{if } x_{k+1} = 1 \texttt{ then } b_{k+1} \\
   &\texttt{else } f_k
   \end{aligned}$$

   If $b_{k+1} = +$, the linear threshold function is "$f_{k+1}(x_1, \ldots, x_{k+1}) = +$ if and only if $x_{k+1} = 1$ or $x_{k+1} = 0 \wedge f_k \geq 1$, if and only if $f_k + 2^k x_{k+1} \geq 1$". If $b_{k+1} = -$, the linear threshold function is just $f_k - 2^k(1 - x_{k+1}) \geq 1$. The weight 2^k is used to let the last term dominates f_k. □

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 $\log_2(l)$.

   (1) When $l = 1$, $r = 0 = \log_2(l)$.
   (2) Let T be the tree with $l > 1$ leaves, and suppose that any tree with $n < l$ leaves has rank at most $\log_2(n)$. Then, any subtrees of T leaves has rank at most $\log_2(l-1)$. (a) If $r_L \neq r_R$, then $r = \max\{r_L, r_R\} \leq \log_2(l-1) < \log_2(l)$ since both the subtrees of T have less than l leaves. (b) Since at least one of the subtrees of T has at most $\frac{l}{2}$ leaves, without loss of generality we can assume the left subtree has at most $\frac{l}{2}$ leaves. If $r_L = r_R$, then the decision tree will have rank $r = r_L + 1 \leq \log_2\left(\frac{l}{2}\right) + 1 = \log_2(l)$. Therefore, by induction, a decision tree with l leaves has rank at most $\log_2(l)$.

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 $O(n^k L)$, where 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.

Let $f \in \mathcal{C}$ be the target function with length L. We will say that f has L levels and the lth level is corresponding to the lth or $(l+1)$th `if-else` statement.

(1) We begin with a lazy version of decision list h

$$\texttt{if } x_1 \vee \overline{x}_1 \vee \cdots \vee x_n \vee \overline{x}_n \texttt{ then } +$$
$$\texttt{else } -$$

that is, we predict $+$ for whatever input.

(2) If h make a mistake on a sample x with a negative label at level l, then we move all variables that predict uncorrectly given x at level l to the next level to output the correct label, and we also change the prediction of the default output in the last level.

(3) When the first level become empty, we just drop this level. This is just for changing the output of the first level of the decision list, and we will be able to do this at most one time.

(4) We repeat (2)-(3) as we recieve more samples.

Any variable in f should never fall below the same level in h produced by this algorithm. We prove this by induction. Without loss of generality, assume that the decision list f is

$$\begin{aligned}
&\texttt{if } \ x_1 = 1 \texttt{ then } b_1\\
&\texttt{else if } x_2 = 1 \texttt{ then } b_2\\
&\qquad\qquad \vdots\\
&\texttt{else if } x_L = 1 \texttt{ then } b_L\\
&\texttt{else } -
\end{aligned}$$

(1) Then $x_1 = 1$ will be in the 1st or 2nd level in h depending on $b_1 = +$ or $b_i = 1$, since we can only make one mistake on this variable.
(2) Assume that for $l \leq k-1$, $x_l = 1$ lies in at most the lth level in h (or $(l+1)$th if $b_1 = -$). For $l = k$, if $x_k = 1$ is at below $(l+1)$th level, then there must be some other variables x_j in the $(l+1)$th level, while in f we first encounter x_k rather than x_j, so x_j will make mistakes since it depends on the result $x_k = 1$ or 0 in f.
In the algorithm, we can move each variable at most L times, from the first level to the $L+1$ level, since the first level may become empty and f has L level (excluding the default one). Also, we have totally $2n$ variables, so we use move at most $2nL$ times, i.e., we will make at most $O(nL)$ mistakes.
To generaize this algorithm for k-decision lists, we just need to set up new variables $\{z_1 \wedge \cdots \wedge z_k | z_1, \ldots, z_k \in \{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}\}$. Now we have at most $(2n)^k$ variables, so the mistake bound would become $O(n^k L)$.

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.)

Proof. A decision tree with l leaves and rank 1 is equivalent to a 1-decision list of length l.
For $k \in \mathbb{N}$ and $k > 1$, suppose that for any decision tree with l leaves and rank $n < k$ can all be transformed to a n-decision list of length at most l. Then for a decision tree T with size s and rank k:

(1) If the rank of its subtrees T_L and T_R are equal, then $\text{rank}(T_L) = \text{rank}(T_R) = k-1$. Then by assumptions, T_L and T_R can be converted to two $(k-1)$-decision lists, denoted by DL_L and DL_R respectively. So DL_L will have the following form:

$$\begin{array}{l}\texttt{if } \; f^L_{k-1,1} \texttt{ then } b^L_1 \\ \texttt{else if } f^L_{k-1,2} \texttt{ then } b^L_2 \\ \qquad\qquad \vdots \\ \texttt{else if } f^L_{k-1,n_L} \texttt{ then } b^L_{n_L} \\ \texttt{else } b^L_{n_L+1}\end{array}$$

where n_L is the number of leaves of T_L, $f^L_{k-1,j}$ and b^L_j are a conjunction of $k-1$ variables and the output in the jth test for $j = 1, \ldots, n_L$, and b_{n_L+1} is the default output. Analogously, DL_R has a similar form. Furthermore, we assume that $b^L_{n_L+1} = b^R_{n_R+1}$ without loss of generality. If this does not hold, we can just modify the last test of DL_R as

$$\texttt{if } \overline{x} \wedge f^R_{k-1,n_R} \texttt{ then } b^R_{n_{R+1}} \texttt{ else } b^R_{n_R}.$$

So we can combine DL_L and DL_R to give a k-decision list for T. Suppose the variable at the node is x, then our k-decision list is given by

$$\begin{array}{l}\texttt{if } x \wedge f^L_{k-1,1} \texttt{ then } b^L_1 \\ \texttt{else if } x \wedge f^L_{k-1,2} \texttt{ then } b^L_2 \\ \qquad\qquad \vdots \\ \texttt{else if } x \wedge f^L_{k-1,n_L} \texttt{ then } b^L_{n_L} \\ \texttt{else if } \overline{x} \wedge f^R_{k-1,1} \texttt{ then } b^R_1 \\ \qquad\qquad \vdots \\ \texttt{else if } \overline{x} \wedge f^R_{k-1,n_R} \texttt{ then } b^R_{n_R} \\ \texttt{else } b^R_{n_R+1}\end{array} \qquad (*)$$

The size of this k-decision is just l, the number of leaves of T.

(2) If $\text{rank}(T_L) < \text{rank}(T_R)$, then $\text{rank}(T_L) < k = \text{rank}(T_R)$. We can first consider a simple case when $\text{rank}(T_{RL}) = \text{rank}(T_{RR}) = k-1$. Same argument in (1) yields the result that T_R can be converted to a k-decision list. Also, since T_L has rank less than k, by assumption it can be converted to a $(k-1)$-decision list (if the number of variables in each test is less than $k-1$, we can just add some 1s). So the following k-decision list is equivalent to T,

$$\begin{array}{l}\texttt{if } x \wedge f^L_{k-1,1} \texttt{ then } b^L_1 \\ \texttt{else if } x \wedge f^L_{k-1,2} \texttt{ then } b^L_2 \\ \qquad\qquad \vdots\end{array}$$

Solution (cont.)

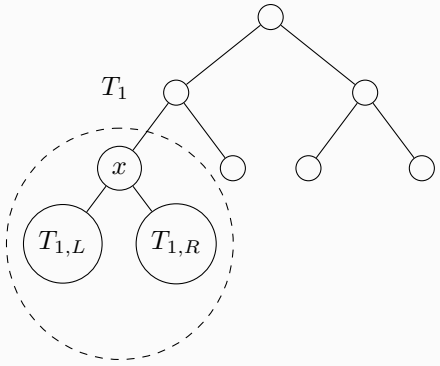
$$\begin{aligned}
&\texttt{else if } x \wedge f^L_{k-1,n_L} \texttt{ then } b^L_{n_L} \\
&\texttt{else if } f^R_{k,1} \texttt{ then } b^R_1 \\
&\qquad\qquad \vdots \\
&\texttt{else if } f^R_{k,n_R} \texttt{ then } b^R_{n_R} \\
&\texttt{else } b^R_{n_R+1}
\end{aligned} \tag{$**$}$$

And the length of it is also l.

For general cases, we start from $T_{1,L}$, any subtree of T. We find its direct ancestor x in T, and denote the subtree with root x by T_1. Then $T_{1,L}$ is just the left subtree of T_1. We know that T_1 is at most rank k, because a tree T with rank k cannot have a subtree with rank larger than k. Without loss of generality, we assume $\text{rank}(T_{1,L}) \geq \text{rank}(T_{1,R})$.

(a) If $\text{rank}(T_{1,L}) = k$, then the right subtree of T_1 will have rank at most $k-1$, because otherwise $\text{rank}(T_1) = k+1$ which is a contradiction. So T_1 can be expressed as a k-decision list like $(**)$.

(b) If $\text{rank}(T_{1,L}) < k$, then both $T_{1,L}$ and $T_{1,R}$ have rank less than k. If $\text{rank}(T_{1,L}) = \text{rank}(T_{1,R}) = k-1$, then $\text{rank}(T_1) = k$ and T_1 can be expressed as a k-decision list like $(*)$. Otherwise $\text{rank}(T_1) < k$, so by assumption it can also be expressed as a $\text{rank}(T_1)$-decision list.



That is, T_1 has rank as most k and can be expressed as a k-decision list of length at most l_{T_1}, the number of leaves of T_1. And we can repeat the above process from bottom to top until we reach the root node. During this process, the length of the decision list does not exceed the number of leaves of the corresponding subtree. As the number of leaves of any subtree of T is less than or equal to l, the number of leaves of T, we know that the length of the final decision list must be at most l. Therefore, if $\text{rank}(T_L) < \text{rank}(T_R)$, then T can be expressed as a k-decision list of length at most l.

(3) The case when $\text{rank}(T_L) > \text{rank}(T_R)$ is analogous.

So by induction, for all $k \in \mathbb{N}$, a decision tree with l leaves and rank k can be expressed as a k-decision list of length at most l.

Thus, from Problem 4, we can learn constant rank decision trees in polynomial time with mistake bound $O(n^k l)$ with $l = \frac{s+1}{2}$ for a binary tree. From Exercise 3, it have rank at most $k = \log_2(l)$. So we need number of $n^{O(\log s)}$ examples. □