Rank-\( r \) decision trees are a subclass of \( r \)-decision lists

Avrim Blum *

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Communicated by M.J. Atallah
Received 20 January 1992
Revised 11 March 1992

Abstract

Blum, A., Rank-\( r \) decision trees are a subclass of \( r \)-decision lists, Information Processing Letters 42 (1992) 183-185.

In this note, we prove that the concept class of rank-\( r \) decision trees (defined by Ehrenfeucht and Haussler) is contained within the class of \( r \)-decision lists (defined by Rivest). Each class is known to be learnable in polynomial time in the PAC model, for constant \( r \). One result of this note, however, is that the simpler algorithm of Rivest can be used for both.

Keywords: Machine learning theory, decision trees, decision lists, analysis of algorithms

1. Introduction

Rivest [5] defines the notion of a decision list as a representation for Boolean functions. He shows that \( k \)-decision lists, a generalization of \( k \)-CNF and \( k \)-DNF formulas, are learnable for constant \( k \) in the PAC (or distribution-free) learning model [8,3]. Ehrenfeucht and Haussler [1] define the notion of the rank of a decision tree, and prove that decision trees of constant rank are also learnable in the PAC model, using a more complicated algorithm. In this note, we prove that any concept (Boolean function) that can be described as a rank-\( r \) decision tree can also be described as an \( r \)-decision list. Thus, the simpler algorithm of Rivest can be used for both cases. Littlestone’s modification of Rivest’s algorithm [4] (generalized by Helmbold, Sloan, and Warmuth [2]) learns decision lists in the more stringent on-line mistake-bound learning model. So, the result given here implies that constant-rank decision trees can be learned in the mistake-bound model as well. In addition, this extends the result of Ehrenfeucht and Haussler that polynomial-size decision trees over \( n \) variables can be learned in time \( O(n^{O(\log n)}) \) from the PAC to the mistake-bound model.

Simon [7] shows that the class of decision trees of rank at most \( r \) over \( n \) variables has VC-dimension \( \sum_{i=0}^{r} \binom{n}{i} \). If only a rough upper bound is needed, then a simpler \( O(n^r) \) bound follows from this note and the known observation that 1-decision lists are a special type of linear separator (and the known VC-dimension of linear separators [9]). Work on learning both constant-rank
decision trees and \( k \)-decision lists in the presence of noise has been done by Sakakibara [6].

### 1.1. Definitions

An example \( \bar{x} \) is a boolean vector \( \{0, 1\}^n \), and we write \( x_i \) to denote the \( i \)th bit of \( \bar{x} \). Let \( V_n \) be a set of \( n \) boolean variables \( v_1, \ldots, v_n \), and define a literal to be either a variable or a negation of a variable. We say example \( \bar{x} \) satisfies variable \( v_i \) if \( x_i = 1 \), and \( \bar{x} \) satisfies \( \overline{v_i} \) if \( x_i = 0 \). A term or monomial is a conjunction of literals; that is, an example satisfies a term if it satisfies all literals in the term.

A decision list is a list of items, each of which is of the form \( \text{term}_i \Rightarrow b_i \), where \( \text{term}_i \) is a monomial and \( b_i \in \{0, 1\} \). The last term in the list must be identically true. The function computed by a decision list \((\text{term}_1 \Rightarrow b_1, \text{term}_2 \Rightarrow b_2, \ldots, \text{term}_m \Rightarrow b_m)\) is as follows. If \( \text{term}_1 \) is satisfied by the example, then the value is \( b_1 \); otherwise, if \( \text{term}_2 \) is satisfied then the value is \( b_2 \), and so forth. A \( k \)-decision list is a decision list where each term contains at most \( k \) literals. The length of a decision list is the number of items.

A decision tree over \( V_n \) is a full binary tree (each internal node has two children), with each internal node labeled with some variable in \( V_n \) and each leaf labeled with "0" or "1". The same variable may appear in multiple internal nodes of tree. A decision tree \( T \) represents a boolean function \( f_T \) over \( \{0, 1\}^n \) defined as follows. If \( T \) is a single leaf with label \( b \in \{0, 1\} \), then \( f_T \) is the constant function \( b \). Otherwise, if \( v_i \) is the label in the root of \( T \), and \( T_0 \) and \( T_1 \) are the left and right subtrees respectively, then \( f_T(x) = f_{T_0}(x) \) if \( x_i = 0 \) and \( f_T(x) = f_{T_1}(x) \) if \( x_i = 1 \).

Ehrenfeucht and Haussler [1] define the rank of a decision tree as follows: If \( T \) is a single leaf, then \( \text{rank}(T) = 0 \). Otherwise, if \( T_0 \) and \( T_1 \) are the left and right subtrees, then

\[
\text{rank}(T) = \begin{cases} 
\max(\text{rank}(T_0), \text{rank}(T_1)) & \text{if } \text{rank}(T_0) \neq \text{rank}(T_1), \\
\text{rank}(T_0) + 1 & \text{otherwise.}
\end{cases}
\]

### 2. The containment theory

Before proving the main theorem, we first note the following simple lemma.

**Lemma 1.** A rank-\( r \) decision tree has some leaf at distance at most \( r \) from the root.

**Proof.** Consider a rank-\( r \) decision tree \( T \). By definition of rank, either the left or right subtree of \( T \) must have rank at most \( r - 1 \). Let us call that subtree \( T' \). Similarly, one of the two subtrees of \( T' \) must have rank at most \( r - 2 \), and so forth. Since a rank-0 decision tree is just a single leaf, this means there must be some leaf with distance at most \( r \) from the root. \( \square \)

So, for example, in a rank-1 decision tree, one of the children of the root must be a leaf; in a rank-2 decision tree, one of the grandchildren of the root must be a leaf.

The basic idea for writing a rank-\( r \) decision tree as an \( r \)-decision list is just as follows. We find a leaf in the decision tree at distance at most \( r \) from the root, and place the literals along the path to the leaf as a monomial at the top of a new decision list. More formally, we prove by induction the following theorem.

**Theorem 2.** For any rank-\( r \) decision tree of \( m \) leaves there exists an equivalent \( r \)-decision list of length at most \( m \).

**Proof.** First, note that a rank-1 decision tree is immediately a 1-decision list, so that is easy. We now argue for general \( r \) by induction on the number of leaves of the decision tree; the base case is handled by the fact that a decision tree of two leaves must have rank 1.

Let \( T \) be the given rank-\( r \) decision tree. There must be some leaf \( l \) at distance at most \( r \) from the root, and let us denote the nodes on the path to \( l \) by \( N_1, N_2, \ldots, N_r \), labeled with variables \( v_{i_1}, \ldots, v_{i_r} \) respectively. Let \( y_1, y_2, \ldots, y_r \) denote
the sequence of literals that must hold true for an example to follow the path to \( l \). For example, if \( l \) is the right child of \( N_r \) then \( Y_r = v_i \), and if \( l \) is the left child then \( Y_r = \overline{v_i} \). Thus, if \( b \in \{0, 1\} \) is the label of \( l \), we know that
\[
y_1 \land y_2 \land \cdots \land y_{r-1} \land y_r \Rightarrow b
\]
in the function defined by \( T \). So, we can put implication (1) at the top of our new \( r \)-decision list, which we will call "\( L \)."

We know that node \( N_r \) has two children in \( T \). Leaf \( l \) is one of them, and let \( N_{r+1} \) be the other (\( N_{r+1} \) may also be a leaf).

We now use the following fact. The decision list \( L \) must be consistent with \( T \). However, if we did not exit at the first line of \( L \), it must be that if "\( y_1 \land \cdots \land y_{r-1} \)" holds, then \( y_r \) must not hold. Thus, (here is the key point) it suffices in creating the decision list after the first line of \( L \) to be consistent with the decision tree \( T' \) obtained by bypassing node \( N_r \) and directly linking \( N_{r-1} \) to \( N_{r+1} \). Now, the decision tree \( T' \) is a tree of rank at most \( r \) which only \( m - 1 \) leaves; we know the rank of \( T' \) is at most \( r \), because the rank of the subtree of \( T \) rooted at node \( N_{r-1} \) cannot be higher than the rank of the subtree rooted at \( N_r \). Thus, by induction, \( T' \) is equivalent to an \( r \)-decision list (or \( r' \)-decision list for \( r' < r \)) \( L' \) of length at most \( m - 1 \). So, we are done: we just output \( L \) as item (1) followed by \( L' \). \( \square \)

Acknowledgment

This work came out of discussions in Ron Rivest’s machine learning theory reading group at MIT. I would like to thank Ron and the members of the reading group for their help in simplifying parts of the argument given here.

References