Lower Bounds on Learning Decision Lists and Trees

Thomas Hancock* Siemens

Tao Jiang† McMaster

Ming Li‡ Waterloo

John Tromp§ Waterloo

Abstract

k-decision lists and decision trees play important roles in learning theory as well as in practical learning systems. k-decision lists generalize classes such as monomials, k-DNF, and k-CNF and like these subclasses is polynomially PAC-learnable [19]. This leaves open the question of whether k-decision lists can be learned as efficiently as k-DNF. We answer this question negatively in a certain sense, thus disproving a claim in a popular textbook [2]. Decision trees, on the other hand, are not even known to be polynomially PAC-learnable, despite their widespread practical application. We will show that decision trees are not likely to be efficiently PAC-learnable. We summarize our specific results.

The following problems cannot be approximated in polynomial time within a factor of $2^{\log^{\delta} n}$ for any $\delta < 1$, unless NP $\subset$ DTIME$[2^{\mathrm{polylog}\, n}]$: a generalized set cover, k-decision lists, k-decision lists by monotone decision lists, and decision trees. Decision lists cannot be approximated in polynomial time within a factor of n^{δ}, for some constant $\delta > 0$, unless NP = P. Also, k-decision lists with l 0-1 alternations cannot be approximated within a factor $\log^{l} n$ unless NP $\subset$ DTIME$[n^{O(\log\log n)}]$ (providing an interesting comparison to the upper bound recently obtained in [1]).

---

*Address: Siemens Corporate Research, 755 College Road East, Princeton, NJ 08540. E-mail: hancock@scr.siemens.com

†Supported in part by NSERC Operating Grant OGP0046613. Address: Department of Computer Science and Systems, McMaster University, Hamilton, Ont. L8S 4K1, Canada. E-mail: jiang@maccs.mcmaster.ca

‡Supported in part by the NSERC Operating Grant OGP0046506 and ITRC. Address: Department of Computer Science, University of Waterloo, Waterloo, Ont. N2L 3G1, Canada. E-mail: mli@math.uwaterloo.ca

§Supported by an NSERC International Fellowship and ITRC. Address: see Li. E-mail: tromp@math.uwaterloo.ca

1 Introduction

This paper proves lower bounds on the approximability of decision lists and trees. An instance of such a problem is a set of boolean n-vectors each labeled with a binary classification, true or false. The desired output is a decision list (or tree) over the n attributes that is consistent with the set of examples and whose size (according to some natural measure) is within a certain approximation factor of the minimum solution size.

This is an important problem in learning theory, since the existence of a polynomial approximation algorithm implies that the class is PAC learnable (given certain technical conditions [6]). Such algorithms, with a mere logarithmic approximation factor, exist for k-DNF and k-CNF, for example. Tighter approximation algorithms imply more efficient learning algorithms (in terms of sample complexity). Furthermore, the non-existence of such algorithms implies negative results for learnability. If there is no polynomial approximation algorithm that produces a representation within a certain factor of the minimum size, then there is no PAC learning algorithm whose hypothesis is within that factor of the minimum possible unless RP = NP [12]. Our main results will show that based on the assumption that NP is not contained in randomized pseudo-polynomial time, size l k-decision lists (or trees) are not learnable by size $l2^{\log^{\delta} l}$ k-decision lists (trees, respectively) for any $\delta < 1$. Under the weaker condition that RP $\neq$ NP, it is hard to learn general decision lists by size $l^{1+\epsilon}$ decision lists for some $\epsilon > 0$.

Decision trees are the key concept in learning systems such as ID3 [15] and CART [7], and have found a use in many applications. There is an enormous amount of machine learning literature concerned with decision trees (see, for example, [14, 15, 16]). However, the PAC-learnability of decision trees remains as one of the most prominent open questions in learning theory. Other than many heuristic learning algorithms using principles such as minimum description length and maximum entropy, the theoretically best learning algorithm [8] PAC-learns a decision tree using $O(n^{\log n})$ time and samples. It is thus a very important question in learning theory (and learning practice) whether decision trees are polynomially PAC-learnable; that is, whether we can find the approximately smallest decision tree with high probability. We will provide a negative result in this direction, showing that decision trees cannot be efficiently approximated. One of the main motivations for the use of decision trees in empirical learning work is that small decision trees are believed to be a good output representation for human comprehensibility. Hence negative results regarding finding such a hypothesis are particularly relevant. Note that it was not even known if finding the smallest decision tree is NP-hard. What was known is that finding a decision tree with the minimum external path length is NP-hard [11, 9].

2 Learning Theory and Complexity of Approximation

This section introduces concepts and results in learning theory and complexity of approximation that are needed in this paper. We will consider the following classes of Boolean functions, with Boolean variables $\{x_1, \ldots, x_n\}$, defined on $\{0, 1\}$:

- M_n: all monomials defined on $\{0, 1\}^n$, where a *monomial* is a conjunction of zero or more literals.

- $M_{n,k}$: all monomials defined on $\{0,1\}^n$ having at most k literals.
- DNF (Disjunctive Normal Form): set of formulas described as disjunctions of monomials defined on $\{0,1\}^n$. CNF (Conjunctive Normal Form) is a conjunction of disjunctions of literals.
- k-DNF: set of the DNF formulas with each monomial termn $M_{n,k}$. k-CNF is defined analogously.
- DL: class of decision lists. A *decision list* defined on $\{0,1\}^n$ is a list $L = (m_1, c_1)(m_2, c_2)\dots(m_l, c_l)$, where for each $i = 1, \dots, l$, $m_i \in M_n$ is a monomial and $c_i \in \{0,1\}$. For an input Boolean vector from $\{0,1\}^n$, the value of list L is c_i if i is the smallest integer such that m_i is satisfied, If no m_i is satisfied then $L = 0$.[1] *Monotone* decision lists are those not containing any negated variables.
- k-DL: class of k-decision lists. A *k-decision list* is a decision list in which each term belongs to $M_{n,k}$.
- DT: class of decision trees. A *decision tree* defined on $\{0,1\}^n$ is a binary tree T, each of whose non-leaf nodes is labeled with one of n variables $x_1, \dots, x_n$, and whose leaves are labeled 0 or 1. The value of T on an input vector is obtained as the label of the leaf reached from the root as follows: at each node go to the left or right child depending on whether the input bit corresponding the label is 0 or 1, respectively.

A PAC learning algorithm, for a hypothesis class H, solves the following problem:

Input: Constants $0 < \epsilon, \delta < 1$, and a source of random examples EX that when queried provides in unit time a pair $< x, c(x) >$ where x is drawn independently according to some unknown distribution D and c is some fixed unknown concept from H.

Output: A hypothesis $h \in H$.

Requirement: The input determines the probability of any hypothesis h being output by the learning algorithm. The essential requirement of PAC learning is that the probability of outputting a *bad* hypothesis (one for which $D(h(x) \neq c(x)) > \epsilon$) is bounded by δ.

The learner is allowed running time polynomial in $1/\epsilon$, $1/\delta$, and the size of the target concept c. For more details of Valiant's PAC-learning model, we refer the reader to [2]. The basic question in learning theory is to identify the polynomially learnable classes. Among the most prominent polynomially learnable classes are monomials, k-DNF, k-CNF, k-CNF, and k-DL. It is clear that monomials, k-DNF, and k-CNF are subclasses of k-DL.

An efficient learning algorithm that produces a hypothesis for a certain class can be converted to a randomized approximation algorithm for that class by a construction of [3] (the essence of the conversion is that approximation for a sample m can be obtained by the learner if the learner is trained with the uniform distribution on m and is forced to achieve accuracy close to $1/|m|$).

Recently, there has been significant progress in the complexity theoretic aspects of approximation. Below we list the results that will be needed in this paper.

[1] Our definition follows [19]. In [2], the class of DL is defined as a more general class such that each term may be a Boolean function from some class K rather than just monomials.

Theorem 1 *Unless* NP = P*:*

1. *Graph Coloring cannot be approximated in polynomial time with ratio* n^ϵ*, for some* $\epsilon > 0$ *[13].*

2. *Set Covering cannot be approximated in polynomial time to within any constant [13].*

3. *Vertex Cover does not have a polynomial time approximation scheme citearora. I.e., for some* $\epsilon > 0$*, it cannot be approximated within a factor* $1 + \epsilon$*.*

Our hardness results for k-decision lists and decision trees start with a basic approximation preserving (linear) reduction from Set Covering to show that approximation within a constant factor is hard. Then we use a "multiplication" process to in essence combine two problems into one in a way that amplifies approximation difficulty. For k-decision lists, this amplification is done using a new form of the set covering problem that we define, show to be difficult to approximate, and then reduce finding a consistent k-decision list to. For decision trees, we do the amplification on decision trees themselves to show how arbitrary approximability can be improved with additional computation. This allows us to achieve a greater lower bound on approximation than the constant implied by our reduction from Set Covering.

Our hardness results for generalized decision lists are based on an approximation preserving reduction from Graph Coloring.

3 Non-approximability of k-Decision Lists

Our results build upon the non-approximability of Set Covering (item 2 above) and Vertex Cover (item 3 above), which can be viewed as a special case of Set Covering. Given a collection $S_1, S_2, \ldots, S_m \subset U = \{1, 2, \ldots, t\}$ of sets, we are to find a minimum-size subcollection that covers U. This lower bound translates to a lower bound on approximability of monomials, where, given a set of positive and negative examples, we are to find a shortest monomial consistent with those examples. The reduction from Set Covering to monomials is as follows. Given an instance of Set Covering, $S_1, S_2, \ldots, S_m \subset \{1, 2, \ldots, t\}$, create examples of length m bits, one bit for each set. For each element of U, create a negative example that has a 0 for all sets containing it and 1s elsewhere. Also create one positive example of all 1s. The latter prevents any negative literals from occurring in a consistent monomial. It is easy to see that a consistent monomial of k literals exists iff a size-k set cover exists. So, assuming NP $\neq$ P, the shortest consistent monomial cannot be approximated in polynomial time within any constant.

Under the stronger assumption NP $\not\subset$ DTIME$[n^{O(\log\log n)}]$, [5, 13] show that Set Covering cannot be approximated in polynomial time with a ratio of $c \log n$, for any $c < 1/8$. (Our results, based on 2, do not benefit from this stronger bound.) For monomials (and k-DNF/k-CNF), this result is sharp, since a greedy algorithm finds a monomial that is at most logarithmically longer than the shortest one [10]. Decision lists, however, are of a different nature, since their individual components are ordered in an essential way. They correspond not to Set Covering, but to a generalization that we call (P, N) *Covering*: Given disjoint sets P, N and a collection $S_1, S_2, \ldots, S_m \subset P \cup N$, we are to find a shortest sequence $S_{i_1}, \ldots, S_{i_k}$ that covers P

such that for each $1 \leq r \leq k$,

$$S'_{i_r} \stackrel{\text{def}}{=} S_{i_r} \setminus \bigcup_{j<r} S_{i_j}$$

is a subset of either P or N. If S'_{i_r} is non-empty and $S'_{i_r} \subset X$, we say S_{i_r} *uncovers* X. Here, X can be P, N, or a subset of either.

3.1 Non-approximability of (P, N) Covering

We show how to amplify the non-approximability of Set Covering by a blow-up reduction from Set Covering to (P, N) Covering. The basis of this blow-up is formed by a construction that "multiplies" a Set Covering instance A with a (P, N) Covering instance B to obtain a (P, N) Covering instance C whose smallest solution has a size close to the product of those of A and B. Let instance A consist of covering sets $s_1, s_2, \ldots, s_m \subset U = \{1, 2, \ldots, t\}$. Let instance B consist of covering sets $r_1, \ldots r_l \subset P \cup N$. Instance C will consist of the following elements: a copy of U, m copies of P and N called $P_i, N_i, 1 \leq i \leq m$ and two extra elements x, y. These elements are partitioned into $P_C = U \cup \bigcup_i N_i \cup \{x\}$ and $N_C = \bigcup_i P_i \cup \{y\}$. Denote the copy of r_j in $P_i \cup N_i$ by $r_{i,j}$. The covering sets of C are

- $r_{i,j}$ for all $1 \leq i \leq m$ and $1 \leq j \leq l$.
- $s_i \cup P_i$, for all $1 \leq i \leq m$.
- $U \cup N_C$
- $P_C \cup N_C$

Claim 2 *If a and b are the sizes of minimum covers of A and B, then the minimum cover of C has size $a(b+1)+2$. Furthermore, for any c, d, given a covering of C of size at most $c(d+1)+2$, one can find either a cover of A of size at most c, or a cover of B of size at most d.*

Proof. Recall that a cover of C means a cover of set P_C. Since this contains an element, x, which occurs in covering set $P_C \cup N_C$ only, set N_C must be covered first. Since only $U \cup N_C$ can uncover $y \in N_C$, set U must be uncovered by the sets $s_i \cup P_i$, which takes exactly a of them. Each of those requires P_i to be covered first, which takes b of the $r_{i,j}$ sets. Thus, $a(b+1)$ sets are necessary and sufficient to cover U, after which the sets $U \cup N_C$ and $P_C \cup N_C$ suffice (and are needed) to complete the covering of P_C. For the second part one can use the induced covering of U or the smallest induced covering of P. Clearly, if the former is of size more than c and the latter of size more than d, then this contradicts the assumption.

We can now define "powers" $A^0, A^1, \ldots$ of a Set Cover instance A, where A^0 is an empty (P, N) Cover instance, and $A^{i+1} = AA^i$. Solving for size, we find that A^i has minimal size $(a^i - 1)\frac{a+2}{a-1}$. Furthermore, Claim 2 implies that, from a cover of A^i of size c, we can find a cover of A of size at most $\sqrt[i]{c\frac{a-1}{a+2}+1}$.

This blow-up technique allows us to prove

Theorem 3 (P, N) *Covering cannot be approximated in polynomial time within a factor of $2^{\log^\delta n}$, for any $\delta < 1$, unless* NP *is contained in* $\mathrm{DTIME}[2^{\mathrm{polylog}\, n}]$.

Proof. Suppose, by way of contradiction, that it could. Let an arbitrary Set Covering instance A have a smallest solution of n. In $2^{\mathrm{polylog}\, n}$ time, we form A^d where $d = \log^r n$ with $r = 2\delta/(1-\delta)$. This (P,N) Covering instance has a smallest solution of size $(n^d - 1)\frac{n+2}{n-1}$. Then, using the (P,N) Covering approximation, we find a solution of size at most $(n^d - 1)\frac{n+2}{n-1}2^{d^\delta \log^\delta n}$, from which we can find a solution to A of size approximately the dth root of that, which is

$$n2^{d^{-(1-\delta)}\log^\delta n} = n2^{\log^{\delta - r(1-\delta)} n} = O(n),$$

which contradicts the non-approximability of Set Covering (Theorem 1). ■

3.2 A Variant of (P,N) Covering and k-decision Lists

We next adapt Theorem 3 to a slight variant of (P,N) Covering, called *Complementary* (P,N) Covering, where the collection of covering sets is closed under complementation. Given an instance A of (P,N) Covering, introduce three new elements x, y, z, let $P' = P \cup \{x\}$ and $N' = N \cup \{y, z\}$. Add for each covering set its complement (which includes x,y and z), and add the complementary covering sets $N \cup \{x, z\}$, $P \cup \{y\}$, $P \cup N \cup \{z\}$, and $\{x, y\}$. to produce a new instance A'. None of $\{x, y, z\}$ can be covered until P is covered, so the complements of original sets are not useful, and once P is covered, it takes two additional sets to cover x and thereby P'. Note that covering N' also takes two additional sets, a fact that will be used later. Hence, A has a size-k solution if and only if A' has a solution of size $k+2$. So, similarly to Theorem 3, we have

Corollary 4 *Complementary (P,N) Covering cannot be approximated in polynomial time within a factor of $2^{\log^\delta n}$, for any $\delta < 1$, unless* NP *is contained in* DTIME$[2^{\mathrm{polylog}\, n}]$.

Using this result we prove the non-approximability of k-decision lists.

Theorem 5 *k-decision lists cannot be approximated in polynomial time by k-decision lists within a factor of $2^{\log^\delta n}$, for any $\delta < 1$, unless* NP *is contained in* DTIME$[2^{\mathrm{polylog}\, n}]$.

Proof. We first show that 1-decision lists are as hard to approximate as Complementary (P,N) Covering, by a proper encoding.

Consider an instance A of Complementary (P,N) Covering: $S_1, \bar{S}_1, \ldots, S_m, \bar{S_m} \subseteq P \cup N$ (we'll assume A is derived as in Section 3.2). For each element s in $P \cup N$ we create a bit-vector e of length m. The ith bit of e is 1 if S_i contains element s, otherwise the ith bit of e is 0. The example e is labeled positive or negative depending on whether s is from P or N, respectively. The 1-decision list problem on these examples is isomorphic to problem A, since a decision list node with literal x_i or $\bar{x}_i$ selects covering set S_i or $\bar{S}_i$, and the constant of the node is then determined by whether this set uncovers P or N. Also, the part of N that remains not covered corresponds to the default 0 classification of examples in the decision list. Thus there is a 1-decision list of l terms iff there is a l set covering for the Complementary (P,N) Covering problem.

To extend the results to k-decision lists, for $k > 1$, we add two additional fields, called x and y, to the bit-vectors. Field x has length k and field y length $k-1$. We create a k-decision list problem instance B as follows. Replace each old example e by $k2^k$ length $m + 2k - 1$ examples exy, consisting of

1. 2^k examples where x ranges over all k bit strings and $y = 1^{k-1}$, all classified the same as the old example.
2. $2^k(k-1)$ examples where x ranges over all k bit strings and y consists of $k-2$ ones and one 0, classified positive or negative as the parity of x is 1 or 0, respectively.

Intuitively, the purpose of these examples is to force $k-1$ of the k literals in each decision-list node to be wasted on distinguishing the examples of type 1 from the examples of type 2, leaving only one literal to select a covering set as in the $k = 1$ case. After all the type 1 examples have been covered in this way, those of type 2 can be covered with 2^{k-1} (a constant) nodes each using all k literals to test x for parity 1.

The next claims conclude the proof of the theorem.

Claim 6 *As long as both positive and negative type 1 examples remain uncovered, each node in L node must test all $k-1$ y variables,*

Proof. If a node doesn't test all $k-1$ y variables, then it will cover some type 2 examples. Consider the first node that covers a type 2 example. If it doesn't test all k x-variables then it will satisfy examples of both parities, a contradiction since these examples were not covered before. If it does test all k variables, then it will satisfy the assumed remaining positive and negative type 1 examples, another contradiction.

Claim 7 *Complementary (P, N) Covering problem A has a size l covering iff there is a solution to the k-decision list problem B of size $l + c_k$, where c_k is a constant depending only on k.*

Proof. Let c_k be the size of a minimum decision list for just the type 2 examples, i.e. parity on k variables (we needn't bother showing that $c_k = 2^{k-1}$). Given a size l solution to A, we can convert each covering set S to a decision list node with monomial $ly_1 \dots y_{k-1}$ where literal l corresponds to S and with constant 1 or 0 according to whether S uncovers P or N. The resulting length l decision list covers all type 1 examples, and can be extended with c_k nodes to cover all type 2 examples. Now for the other direction, assume we have a decision list L of size $l + c_k$ that is a solution to B.

Claim 6 implies that a prefix of L corresponds to a covering of P or N. As noted in Section 3.2, a covering of N implies a same size covering of P. Since the remainder of L must therefore be a decision list for all the remaining type 2 examples, it must have length at least c_k and hence the prefix has length at most l. ■ (End of Proof of Theorem 5.)

The following learning result follows from the standard technique mentioned in the introduction [12].

Corollary 8 *For no $\delta < 1$ does there exist a polynomial time PAC learning algorithm for k-decision lists whose hypothesis class is k-decision lists of size no more than $l2^{\log^\delta l}$, where l is the size of the target k-decision list, unless* $\mathrm{NP} \subset \mathrm{RTIME}[2^{\mathrm{polylog}\ n}]$.

3.3 Non-approximability of Monotone 1-decision Lists by Monotone Decision Lists

In learning theory, one often considers the problem of learning a class of concepts by a *broader* class of hypothesis. The freedom to formulate hypothesis from a richer class generally makes the learning task easier. Analogously, we can ask whether it is for instance easier to approximate a k decision list with a $2k$ decision list, or even an arbitrary decision list, with no bound on the length of the monomials. We provide an answer to a variation of the latter problem, where no negative literals are allowed in the monomials.

Theorem 9 *Monotone* 1*-decision lists cannot be approximated in polynomial time by monotone decision lists within a factor of* $2^{\log^\delta n}$*, for any* $\delta < 1$*, unless* NP *is contained in* $\mathrm{DTIME}[2^{\mathrm{polylog}\, n}]$.

Proof. We present a variant on the multiplication of Section 3.1 to produce an instance of (P, N) Covering where the intersections of covering sets are of no use. This variant is based on the Vertex Cover problem, which can be regarded as a restricted form of Set Covering where each element appears in exactly 2 covering sets.

Again the basis of the blow-up is formed by a construction that multiplies a Vertex Covering instance A with a (P, N) Covering instance B to obtain a (P, N) Covering instance C whose smallest solution has a size close to the product of those of A and B. Let instance A consist of covering sets $s_1, s_2, \ldots, s_m \subset U = \{1, 2, \ldots, t\}$. Let instance B consist of covering sets $r_1, \ldots r_l \subset P \cup N$. Instance C will consist of the following elements: a copy of U, $m(m+1)/2$ copies of P and N indexed by an unordered pair of the m set indices ($P_{i,j} = P_{j,i}$ and $N_{i,j} = N_{j,i}$), and two extra elements x, y. We take $P_C = U \cup \bigcup_{1 \le i,j \le m} N_{i,j} \cup \{x\}$ and $N_C = \bigcup_{1 \le i,j \le m} P_{i,j} \cup \{y\}$. The copy of r_h in $P_{i,j} \cup N_{i,j}$ is referred to as $r_{i,j,h}$. We take the following covering sets:

- for each $1 \le i \le m, 1 \le h \le l$ a set $S_{i,h} = \cup_{1 \le j \le m} r_{i,j,h}$
- for each $1 \le i \le m$, a set $S_i = s_i \cup \bigcup_{1 \le j \le m} P_{i,j}$
- for each $1 \le i, j \le m$, a set $I_{i,j} = S_i \cap S_j = (s_i \cap s_j) \cup P_{i,j}$
- $U \cup N_C$
- $P_C \cup N_C$

Note that the collection of sets containing parts of both P_C and N_C is closed under intersections since for any i, $S_i \subset U \cup N_C \subset P_C \cup N_C$. Also, the intersection of 3 or more sets S_i is empty, because in the Vertex Cover problem, each element appears in only 2 covering sets. The sets $S_{i,a}$ are not closed under intersections, but such intersections are not helpful in a covering anyway.

Claim 10 *If a and b are the sizes of minimum covers of A and B, then the minimum cover of C has size $a(b+1)+2$. Furthermore, for any c, d, given a covering of C of size at most $c(d+1)+2$, one can find either a cover of A of size at most c, or a cover of B of size at most d.*

Proof. Since x occurs in covering set $P_C \cup N_C$ only, set N_C must be covered first. Since only $U \cup N_C$ can uncover $\{y\}$, set U must be uncovered by the sets S_i, which takes exactly a of them. Using $I_{i,j}$ gives no advantage over using either S_i or S_j since $P_{i,j}$ is as hard to cover as $\cup_k P_{i,k}$ or $\cup_k P_{j,k}$. Each of the a S_i sets requires $P_{i,i}$ to be covered first, which takes b of the $r_{i,h}$ sets. Thus, $a(b+1)$ sets are necessary and sufficient to cover U, after which the sets $U \cup N_C$ and $P_C \cup N_C$ suffice to complete the covering of P_C. These two sets are also necessary to cover x, For the second part one can use the induced covering of U or the smallest induced covering of P. Clearly, if the former is of size more than c and the latter of size more than d, then this contradicts the assumption. ■

As before, we can define powers of a Vertex Cover instance A, and find that A^i has minimal size $(a^i - 1)\frac{a+2}{a-1}$.

We continue the proof of Theorem 9. Suppose, by way of contradiction, that monotone 1-decision lists can be approximated by monotone decision lists within a factor of $2^{\log^\delta n}$ for some $\delta < 1$. Let an arbitrary Vertex Cover instance have a smallest solution of size n, and let ϵ be any positive number. Let $d = \log^r n$, where $r = 2\delta/(1-\delta)$, and consider the dth power of the Vertex Cover problem. The standard encoding of Theorem 5 yields an instance of the monotone decision list problem, any solution of which is essentially a monotone 1-decision list, since intersections of covering sets are not useful. Hence the approximability implies we can find a solution to the blown-up Vertex Cover problem of size $n^d 2^{d^\delta \log^\delta n}$, from which we can find a Vertex Cover solution of size approximately the dth root of that, which is

$$n2^{d^{-(1-\delta)} \log^\delta n} = n2^{\log^{\delta - r(1-\delta)} n} \le n2^{\log^{-\delta} n} \le n(1+\epsilon),$$

for any fixed $\epsilon > 0$ and sufficiently large n. This contradicts the non-approximability of Vertex Cover (Theorem 1). ■

4 Non-approximability of General Decision Lists

In this section, we will show that decision lists also cannot be efficiently approximated, unless NP = P. Notice that the non-approximability results for k-DL in the last section does not say anything about DL, and our bound in this section will be stronger than that of the previous section.

The result below on DNF easily follows from a reduction from the Graph Coloring problem to the approximation of the smallest consistent DNF given in [18] (see also [12]) and the n^ϵ non-approximability lower bound for Graph Coloring in [13].

Theorem 11 [Pitt-Valiant, Lund-Yannakakis] *DNF's cannot be approximated in polynomial time by DNF's within a factor of n^ϵ, for some $\epsilon > 0$, unless* NP = P.

We will prove the same non-approximability ratio for decision lists. Our result in fact strengthens Theorem 11 by showing that DNF's cannot be approximated by the richer class of decision lists within a ratio of n^ϵ.

Theorem 12 *Decision lists cannot be approximated in polynomial time by decision lists within a factor of* n^{ϵ}*, for some* $\epsilon > 0$*, unless* NP $=$ P.

Proof. We use a reduction similar to that of [18], which linearly reduces the Graph Coloring problem to DNF, although our reduction is slightly more involved.

Given a graph $G = (V, E)$ on n nodes and m edges, we reduce it to an instance of decision lists. Construct positive examples

$$1^{i-1}01^{n-i} \tag{1}$$

(0 in position i, 1s elsewhere), one for each vertex, and negative examples

$$1^{i-1}01^{j-i-1}01^{n-j} \tag{2}$$

(0s in positions i, j, 1s elsewhere), one for each edge $\{i, j\}$. We will show that G is k colorable iff there is a decision list of k terms, for any k.

If G is k colorable, then [18] shows that there is a DNF formula of k terms, which is trivially expressed as a decision list of k terms. It remains to show that if there is a decision list of k terms, then we can find a k-coloring of G. This direction needs a nontrivial proof different from [18].

Consider a decision list $L = (m_1, b_1)(m_2, b_2)\ldots(m_k, b_k)$ of k terms consistent with these positive and negative examples in (1) and (2). We will produce a k-coloring of G from L.

Say that c satisfies a vertex or edge if m_c satisfies the corresponding example. Say that c uncovers a vertex or edge if c satisfies it and no $d < c$ does so. If $b_c = 1$, then by consistency of L, c only uncovers vertices, while if $b_c = 0$, it only uncovers edges.

Convert G to a directed graph by directing all edges in G arbitrarily. The color assigned to a vertex i is the minimal c that either

1. uncovers i, or

2. uncovers exactly one edge, where that edge is directed to i.

Notice that in the second case, color c is given to just one vertex, i.

Now we will show that no pair of neighboring vertices get the same color. Consider an edge $e = (i, j)$, directed from i to j. Suppose, by way of contradiction, that vertices i and j are assigned the same color c. Then $b_c = 1$, because otherwise the color c is assigned to at most one vertex according to item 2 above. Hence, c must satisfy both i and j; that is, m_c satisfies the positive examples $1^{i-1}01^{n-i}$ and $1^{j-1}01^{n-j}$ corresponding to i and j. Hence the monomial m_c does not include literals x_i, x_j, or any negative literals. But this means m_c satisfies e. So by consistency of L, e must have been uncovered before, say, by d. But in order to satisfy $e = (i, j)$, m_d may contain neither x_i nor x_j nor any negative literals $\bar{x_k}$, $k \neq i, j$. This leads to a contradiction with i and j being colored c, since either

- d satisfies vertex i or j in which case i or j must have been uncovered by some smaller d', hence *not* colored c,

- or m_d includes both $\bar{x_i}$ and $\bar{x_j}$, in which case only edge e is satisfied by d. But in this case, by our rule 2, we would have colored j with color d rather than c.

Therefore, no pair of vertices connected by an edge are colored by the same color. Furthermore, all vertices are uncovered by some c. We have produced a k coloring for the graph G.

Since the above reduction preserves minimum solution size, approximation of decision lists within a factor of n^ϵ would imply the same approximation of Graph Coloring, and hence is impossible under the given assumptions. ■

The above proof also shows

Corollary 13 *DNF's cannot be approximated in polynomial time by decision lists within a factor of* n^ϵ, *for some* $\epsilon > 0$, *unless* NP = P.

Corollary 14 *There exists an* $\epsilon > 0$ *such that there is no polynomial time PAC learning algorithm for decision lists (or DNF) that, when run on examples from a decision list (DNF, respectively) of size* l, *will on success (i.e. with probability exceeding* $1-\delta$*) always output as a hypothesis a decision list of size at most* $l^{1+\epsilon}$, *unless* NP = RP.

5 Non-approximability of Decision Trees

We now consider the problem of finding a small decision tree consistent with a set M of examples. For notational convenience we measure the size of the tree as the number of leaves (*i.e.*, one more than the number of interior nodes in a binary tree).

Theorem 15 *Unless* P = NP, *there is a constant* $c > 1$ *such that decision trees cannot be approximated in polynomial time by decision trees within a factor* c.

Proof. The argument is entirely similar to the one used in section 3 to show the hardness of approximating monomials. Decision trees are clearly at least as expressive as monomials. It remains to show that for the given boolean vectors (a negative one for each element of the universe and an all 1s positive one), they are not more expressive. To this end, observe that in a consistent decision tree, any node's 0 subtree can be safely replaced by a single lead labelled 0, the result being a degenerate tree that's equivalent to a monomial. ■

We now discuss how to amplify this result to show that an algorithm to approximate decision trees within a factor of $2^{\log^\delta n}$ could in fact be improved to approximate within an arbitrary constant factor in pseudo-polynomial time.

A key idea is that we can "square" a decision tree problem as described in the following construction. Let M be a set of m examples with n variables. Construct a set M' with m^2 examples over $2n$ variables by concatenating xy for each $x, y \in M$. If $f(x)$ is the classification of example x, let each example xy have the classification $f'(xy) = f(x) \oplus f(y)$ (i.e. f' is the exclusive-OR of two copies of f over different variables).

The obvious way to build a decision tree for M' is to take a decision tree T for M on the first n attributes and place at each of its leaves a copy of T over the second set of n attributes. This converts a k-leaf tree for M into a k^2 leaf tree for M'. We show that this construction is optimal as a consequence of the following stronger lemma which shows how we can efficiently convert a tree for M' back to trees for M in a manner that has implications for approximation.

Lemma 16 *There is an efficient algorithm to convert a decision tree with k' leaves that is consistent with M' into a tree with no more than $\sqrt{k'}$ leaves that is consistent with M.*

Proof. Let T' be such a k' leaf decision tree consistent with M'. We use the notation that $x_1, \ldots, x_n, y_1, \ldots, y_n$ are the variables in T', where the x_i's test the first n attributes and the y_i's test the second n.

Note that if examples xy and $x'y'$ (for $x, y, x', y' \in M$) both reach the same leaf in T', then examples xy' and $x'y$ must also reach that leaf. To avoid contradiction it must be true that x and x' have the same classification in M and likewise for y and y'. More generally, the path to a leaf in T' must determine sufficient information to determine two classifications for each example that reaches the leaf - one based on the x_i's and one based on the y_i's (the leaf's label is the XOR of those two classifications).

To prove the Lemma we first consider the simple case where T' is constructed such that every path in T' has all its tests of the first n attributes (variables $x_1, \ldots, x_n$) before it tests any of the second n attributes (variables $y_1, \ldots, y_n$). We describe such a tree as being in "standard form". The top portion of T' that tests $x_1, \ldots, x_n$ must have k distinct paths, since otherwise we could form a fewer than k leaf tree for M by taking this top portion and giving each path a leaf value corresponding to the unique classification of all the x's in M that follow that path (the previous paragraph states that these tests suffice to assign a single class consistently with M). Similarly, each of the subtrees over $y_1, \ldots, y_n$ must also have k or more leaves. This implies both that T' has at least $k' = k^2$ leaves and also that we can derive from T' a $\sqrt{k'}$ leaf tree for M, simply by taking whichever is smallest of the $x_1, \ldots, x_n$ subtree or any of the $y_1, \ldots, y_n$ subtrees.

We now argue that if T' is not in standard form, then an efficient transformation can find another equivalent tree that is in standard form and has no more leaves than T'. From this the lemma follows.

The transformation process will process T' incrementally from the bottom up. After each node is processed, the following two properties will be true:

1. The subtree rooted at the node is in standard form.

2. All of the various $y_1, \ldots, y_n$ subtrees reachable from the node are identical (except possibly for having complementary leaf settings).

Both properties are trivially true at the leaves. We show how to process a node both of whose children have been processed in such a way as to achieve the two properties (without increasing the size of the tree or changing the function it represents).

We consider two cases. First, suppose the node being processed tests a variable x_i ($1 \leq i \leq n$). In this case property 1) is already true. To achieve property 2), note that each of the $y_1, \ldots, y_n$ subtrees reachable from this node must perform exactly enough tests to uniquely decide on the y_i classification of each example that reaches it. These subtrees get the same set of examples over $y_1, \ldots, y_n$, and so they need only differ on the value of their leaves (which are the XOR of whatever the x and y classifications turn out to be). Thus we can choose the smallest of all the x_i node's y_i subtrees to replicate onto every path (thus achieving property 2) while ensuring that T' does not grow.)

The second case is when the node being processed tests variable y_i ($1 \leq i \leq n$). Assume w.l.o.g. that the y_i node's smaller subtree is its 0-subtree. Then transform T' by moving y_i below tests of $x_1, \ldots, x_n$ as follows. Replace this test of y_i with its (smaller) 0-subtree and then reinsert y_i on all the edges leading from a variable x_j to a variable y_k. Each such new test of y_i will have as it's 1-subtree the tree over $y_1, \ldots, y_n$ that appeared throughout y_i's original 1-subtree (with the leaves set to the appropriate XOR value). It's not hard to verify that this transformed tree is no larger than T', is equivalent, and satisfies the two properties.

This process iterates up T', and once the root is processed we are done. ■

The construction generalizes to an arbitrary depth d. I.e. by concatenating d copies of the original variables we get a sample $M^{(d)}$ of m^d examples over dn attributes for which the minimum decision tree has k^d nodes iff k is the minimum number of nodes for a decision tree over M. Entirely analogous to theorem 3, this blow-up technique gives us

Theorem 17 *Decision trees cannot be approximated in polynomial time by decision trees within a factor of* $2^{\log^\delta n}$, *for any* $\delta < 1$, *unless* NP *is contained in* DTIME$[2^{\mathrm{polylog}\, n}]$.

Proof. Suppose, by way of contradiction, that it could. Let an arbitrary Set Covering instance A have a smallest solution of n. In $2^{\mathrm{polylog}\, n}$ time, we form a tree decision problem A^d where $d = \log^r n$ with $r = 2\delta/(1-\delta)$. The smallest solution of A' has size n^d. Then, using the tree decision approximation, we find a solution of size at most $n^d 2^{d^\delta \log^\delta n}$, from which we can find a solution to A of size the dth root of that, which is

$$n2^{d^{-(1-\delta)} \log^\delta n} = n2^{\log^{\delta - r(1-\delta)} n} = O(n),$$

which contradicts the non-approximability of Set Covering (Theorem 1). ■

Corollary 18 *For no* $\delta < 1$ *does there exist a polynomial time PAC learning algorithm for decision trees whose hypothesis class is decision trees with no more than* $l2^{\log^\delta l}$ *leaves, where* l *is the number of leaves in the target decision tree, unless* NP $\subset$ RTIME$[2^{\mathrm{polylog}\, n}]$.

6 Acknowledgments

We thank Martin Anthony and Ron Rivest for information about decision lists. Also thanks to Lisa Hellerstein for pointing out an omission in our proofs.

References

[1] Aditi Dhagat and Lisa Hellerstein. PAC learning with irrelevant attributes, *Proc. 35rd IEEE Symp. Found. Comp. Sci.*, 1994, 64–74.

[2] M. Anthony and N. Biggs. *Computational Learning Theory.* Cambridge University Press, 1992.

[3] R. Board and L. Pitt, On the necessity of Occam Algorithms. 1990 *STOC*, pp. 54-63.

[4] A. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof Verification and Hardness of Approximation Problems. *Proc. 33rd IEEE Symp. Found. Comp. Sci.*, 1992, 14–23.

[5] M. Bellare, S. Goldwasser, C. Lund, and A. Russel. Efficient probabilistically checkable proofs and applications to approximation. *Proc. 25th ACM Symp. on Theory of Computing*, 1993, 294–304.

[6] A. Blumer and A. Ehrenfeucht and D. Haussler and M. Warmuth. Learnability and the Vapnik-Chervonenkis Dimension. *J. Assoc. Comput. Mach.*, 35(1989), 929-965.

[7] L. Breiman, J. Friedman, R. Olshen and C. Stone. Classification and Regression Trees. Wadsworth International Group, Belmont, CA (1984).

[8] A. Ehrenfeucht and D. Haussler. Learning decision trees from random examples. *COLT'88.*

[9] M. Garey and D. Johnson. *Computers and Intractability.* Freeman, New York, 1979.

[10] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence* 36:2(1988), 177–222.

[11] L. Hyafil and R. Rivest. Constructing optimal decision trees is NP-complete. *IPL*, 5:1(1976), 15-17.

[12] M. Kearns, M. Li, L. Pitt, L. Valiant. On the learnability of Boolean functions. *Proc. 19th STOC*, 1987, New York, 285–295.

[13] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Proc. 25th ACM Symp. on Theory of Computing*, 1993, 286–293.

[14] J. Mingers. An Empirical Comparison of Selection Measures for Decision-Tree Induction. *Machine Learning.* 3 (1989), 319–342.

[15] J.R. Quinlan. Induction of decision trees. *Machine Learning.* 1 (1986), 81–106.

[16] J.R. Quinlan and R. Rivest. Inferring decision trees using the minimum description length principle. *Inform. Computation*, 80(1989), 227–248.

[17] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. Extended abstract in *Proc. 20th ACM Symp. on Theory of Computing.* 1988, 229-234; full version in *Journal of Computer and System Sciences* 43, 1991, 425–440.

[18] L. Pitt and L. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35:4(1988), 965–984.

[19] R. Rivest. Learning decision lists. *Machine Learning*, 2(1987), 229–246.

[20] L. Valiant. A theory of the learnable. *Communications of the ACM*. 27(1984), 1134–1142.