

Decision lists

Martin Anthony

A chapter for *Boolean Methods and Models* (ed. Yves Crama and Peter L. Hammer)

Contents

1	Introduction	3
2	Decision lists	3
2.1	Definition	3
2.2	Special types of decision list	5
3	Representation of Boolean functions as decision lists	7
3.1	DNF and decision list representations	7
3.2	Universality of positive-term decision lists	10
4	Algorithmic aspects of decision lists	11
4.1	Membership problems	11

4.2	Extending and learning decision lists	12
4.3	Algorithmic issues for positive-term decision lists	15
5	Properties of 1-decision lists	16
5.1	Threshold functions and 1-decision lists	16
5.2	Characterizations of 1-decision lists	18
6	Threshold decision lists	20
6.1	Definition and geometrical interpretation	20
6.2	Algorithmics and heuristics of the chopping procedure	21
7	Threshold network representations	22
8	Representational power of threshold decision lists	23
8.1	A function with long threshold decision list representation	23
8.2	Multi-level threshold functions	24
9	Conclusions	25

1 Introduction

Decision lists provide a useful way of representing Boolean functions. Just as every Boolean function can be represented by a DNF formula, we shall see that every Boolean function can also be represented by a decision list. This representation is sometimes more compact. By placing restrictions on the type of decision list considered, we obtain some interesting subclasses of Boolean functions. As we shall see, these subclasses have some interesting properties, and certain algorithmic questions can be settled for them.

2 Decision lists

2.1 Definition

Suppose that K be any set of Boolean functions on $\{0, 1\}^n$, n fixed. We shall usually suppose (for the sake of simplicity) that K contains the identically-1 function $\mathbf{1}$. A Boolean function f with the same domain as K is said to be a *decision list* based on K if it can be evaluated as follows. Given an example y , we first evaluate $f_1(y)$ for some fixed $f_1 \in K$. If $f_1(y) = 1$, we assign a fixed value c_1 (either 0 or 1) to $f(y)$; if not, we evaluate $f_2(y)$ for a fixed $f_2 \in K$, and if $f_2(y) = 1$ we set $f(y) = c_2$, otherwise we evaluate $f_3(y)$, and so on. If y fails to satisfy any f_i then $f(y)$ is given the default value 0.

The evaluation of a decision list f can therefore be thought of as a sequence of ‘if then else’ commands:

```
if  $f_1(y) = 1$  then set  $f(y) = c_1$ 
    else if  $f_2(y) = 1$  then set  $f(y) = c_2$ 
        ...
        ...
        else if  $f_r(y) = 1$  then set  $f(y) = c_r$ 
            else set  $f(y) = 0$ .
```

We define $DL(K)$, the class of *decision lists based on K* , to be the set of finite sequences

$$f = (f_1, c_1), (f_2, c_2), \dots, (f_r, c_r),$$

such that $f_i \in K$ and $c_i \in \{0, 1\}$ for $1 \leq i \leq r$. The values of f are defined by $f(y) = c_j$ where $j = \min\{i \mid f_i(y) = 1\}$, or 0 if there are no j such that $f_j(y) = 1$. We call each f_j a *test* (or, following Krause [15], a *query*) and the pair (f_j, c_j) a *term* of the decision list.

Decision lists were introduced by Rivest [19], where a key concern was to develop a learning algorithm for them. (This is discussed later in this chapter.)

Note that we do not always draw a strong distinction between a decision list as a Boolean function, and a decision list as a *representation* of a Boolean function. Strictly speaking, of course, a decision list is a representation of a Boolean function, just as a DNF formula is.

There is no loss of generality in requiring that all tests f_i occurring in a decision list are distinct. This observation enables us to obtain the following bound:

$$|DL(K)| \leq \sum_{i=0}^{|K|} \binom{|K|}{i} i! 2^i \leq 2^{|K|} |K|! \sum_{i=0}^{|K|} \frac{1}{(|K| - i)!} = 2^{|K|} |K|! \sum_{i=0}^{|K|} \frac{1}{i!} \leq e 2^{|K|} |K|!.$$

(Each decision list of length i is formed by choosing i functions of K , in a particular order, and assigning a $c_i \in \{0, 1\}$ to each.)

Example: Suppose that $K = M_{3,2}$, the set of monomials (that is, simple conjunctions or terms) of length at most two in three Boolean variables. Consider the decision list

$$(x_2, 1), (x_1 \bar{x}_3, 0), (\bar{x}_1, 1).$$

Those examples for which x_2 is satisfied are assigned the value 1: these are 010, 011, 110, 111. Next the remaining examples for which $x_1 \bar{x}_3$ is satisfied are assigned the value 0: the only such example is 100. Finally, the remaining examples for which \bar{x}_1 is satisfied are assigned the value 1: this accounts for 000 and 001, leaving only the example 101 which is assigned the value 0. \square

Suppose that $K = M_{n,k}$ is the set of monomials (or terms) consisting of at most k literals, so each test is a simple conjunction of degree at most k . Then, following Rivest [19], $DL(K)$ is usually denoted k -DL and we call such decision lists *k-decision lists*. (Krause [15] defines a k -decision list to be one in which each test involves at most k variables, but such a decision list can be transformed into one in which the tests are in $M_{n,k}$.)

Note that when $K = M_{n,k}$, we have $|K| \leq (2n)^k$ and hence

$$|k\text{-DL}| = |DL(M_{n,k})| \leq 2^{(2n)^k} e ((2n)^k)! = 2^{O(n^k \log n)},$$

for fixed k .

Later, we will want to consider K being the set of threshold functions, but unless it is explicitly said so, K will either be $M_{n,k}$ for some fixed k , or simply M_n , the set of all monomials on n variables.

2.2 Special types of decision list

By restricting the types of decision list considered, subclasses of decision list arise. We have already witnessed this, when we considered the k -decision lists; these arise from restricting the degree of each test in the decision list to be no more than k . Later in this chapter we shall look more closely at the very special case in which $k = 1$.

Rather than restrict the degree of each individual test, a restriction could be placed on the total number of terms in the decision list: an r -term decision list is (a function representable by) a decision list in which the number of terms is no more than r . We can also combine structural restrictions on decision lists. For example, the r -term k -DLs are those k -DLs in which the number of terms is at most r . (So, here, there is a restriction both on the number of terms, and on the degree of each test.)

As observed by Guijarro *et al.* [10], any function representable by a decision list with few terms (but possibly high degree) is also representable by one with terms of low degree (but possibly many terms).

Theorem 2.1 (Guijarro *et al.* [10]) *Suppose that $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is (representable by) a decision list with r terms. Then f is also (representable by) an r -decision list.*

Proof: Suppose the decision list

$$f = (f_1, c_1), (f_2, c_2), \dots, (f_r, c_r)$$

is given, where, as we may assume, $c_r = 1$. We construct an r -decision list g representing the same function. First, for each choice of a literal from each of f_1, \dots, f_r , we have a term of g of the form $(T, 0)$, where T is the conjunction of the negations of these literals. We take all such terms, in any order, as the first set of terms of g . Note that each such T is of degree no more than r . For example, if f is the decision list

$$(x_2, 1), (\bar{x}_1 x_4, 0), (x_1 x_3, 1),$$

then take the first four terms of g , in any order, to be

$$(\bar{x}_2 x_1 \bar{x}_1, 0), (\bar{x}_2 \bar{x}_4 \bar{x}_1, 0), (\bar{x}_2 x_1 \bar{x}_3, 0), (\bar{x}_2 \bar{x}_4 \bar{x}_3, 0).$$

(In fact, in this case the first term is vacuous and can be deleted.) Next, we consider in turn each of the terms (f_i, c_i) as i is decreased from r to 2. Corresponding to f_i , we form terms (T, c_i) of g by choosing a literal from each preceding term f_1, \dots, f_{i-1} in f , and forming the conjunction of the negations of these. (Note that these tests have degree no more than $i - 1$, which is less than r .) If $c_1 = 0$ we are then done; otherwise, we add $(\mathbf{1}, 1)$ as the last term of g . For example, for the example decision list, a final suitable g is as follows:

$$(\bar{x}_2 \bar{x}_4 \bar{x}_1, 0), (\bar{x}_2 x_1 \bar{x}_3, 0), (\bar{x}_2 \bar{x}_4 \bar{x}_3, 0), (\bar{x}_2 x_1, 1), (\bar{x}_2 \bar{x}_4, 1), (\bar{x}_2, 0), (\mathbf{1}, 1).$$

(We have deleted the redundant first term created above). □

Additionally, Bshouty [5] has obtained the following result, which gives a better dependence on r (at the expense of some dependence on n).

Theorem 2.2 (Bshouty [5]) *Suppose that $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is (representable by) a decision list with r terms. Then f is also (representable by) a k -decision list, where $k = 4\sqrt{n \ln n \ln(r + 1)}$.*

The proof of Bshouty's theorem (which is omitted here) relates decision lists to certain type of decision tree (in which the leaves are k -decision lists).

Another special type of decision list arises when the tests are required to be positive monomials. Guijarro *et al.* [10] refer to such decision lists as *monotone term decision lists*. Here, we shall instead call them *positive-term decision lists*. Note that, even though

the tests are positive, the overall function computed by the decision list need not be positive. Guijarro *et al.* studied a number of aspects of this class, and, as we shall see later, discovered that there are efficient algorithms for many problems associated with the class.

3 Representation of Boolean functions as decision lists

3.1 DNF and decision list representations

We first state a relationship between k -decision lists and special classes of Boolean functions. For any $1 \leq n$, k -DNF denotes the Boolean functions which have a DNF formula in which each term is of degree at most k ; dually, k -CNF denotes the set of functions having a CNF representation in which each clause involves at most k literals.

The following result, noted by Rivest [19], is easily obtained.

Theorem 3.1 *Let K be any set of Boolean functions. The disjunction of any set of functions in K is a decision list based on K . Explicitly, $f_1 \vee f_2 \vee \dots \vee f_r$ is represented by the decision list*

$$(f_1, 1), (f_2, 1), \dots, (f_r, 1).$$

It follows immediately from this that any k -DNF function, as the disjunction of terms of degree at most k , is also a k -decision list. It is easy to see, however, that, for $0 < k < n$, there are k -decision lists that are not k -DNF functions. For example, the function f with formula $x_1 x_2 \dots x_n$ is certainly not a k -DNF. (This is quite apparent: it has just one true point, whereas any k -DNF has at least $2^{n-k} \geq 2$ true points, since any one of its terms does.) However, f can be expressed as the following 1-decision list:

$$(\bar{x}_1, 0), (\bar{x}_2, 0), \dots, (\bar{x}_n, 0), (\mathbf{1}, 1).$$

If K contains the identically-1 function $\mathbf{1}$, then $DL(K)$ is closed under complementation, since

$$(f_1, 1 - c_1), (f_2, 1 - c_2), \dots, (f_r, 1 - c_r), (\mathbf{1}, 1)$$

is the complement of

$$(f_1, c_1), (f_2, c_2), \dots, (f_r, c_r).$$

Thus, in contrast to the DNF and CNF representations, the decision list representations of a function and its negation are of the same size (but, possibly, for a difference of one additional term).

In particular, since k -CNF functions are the complements of k -DNF functions, and since k -DL contains k -DNF, we have that k -DL contains k -CNF also. (Here, by identifying it as a monomial with no literals, we use also the fact that the identically-one function belongs to $K = M_{n,k}$.) In fact, we have the following result, due to Rivest [19]. The fact that the containment is strict demonstrates that the k -decision list representation is, in fact, more powerful than k -DNF and k -CNF representations.

Theorem 3.2 (Rivest [19]) *For $n \geq 2$ and $k \geq 1$,*

$$k\text{-DNF} \cup k\text{-CNF} \subseteq k\text{-DL},$$

and the containment is strict for $n > 2$ and $0 < k < n$.

Proof: The containment has been established in the arguments just given. It remains to show that the containment is strict. We use the fact that if a Boolean function has a prime implicant of degree s , then it does not belong to k -DNF for any $k < s$. We deal first with the case $k = 1$. Consider the function f represented by the 1-decision list

$$(x_1, 0), (x_2, 1), (x_3, 1).$$

Since f has \bar{x}_1x_2 as a prime implicant, it is not in 1-DNF, and since the complement \bar{f} has $\bar{x}_2\bar{x}_3$ as an implicant, \bar{f} is not in 1-DNF, and hence f is not in 1-CNF. Now suppose $n > 2$, that $1 < k < n$ and that k is odd. (The case of even k can be treated similarly.) Let g_k denote the parity function on the first k variables x_1, x_2, \dots, x_k (that is, the exclusive-or of them), regarded as a function on $\{0, 1\}^n$. Through its DNF representation, g_k can be represented by a k -decision list, ℓ . Consider the function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ represented by the k -decision list

$$(\bar{x}_1x_{k+1}, 0), (x_1x_{k+1}, 1), \ell.$$

Then, f has degree- $(k + 1)$ prime implicant $x_1\bar{x}_2\bar{x}_3 \dots \bar{x}_k\bar{x}_{k+1}$, and so is not in k -DNF. Furthermore, the complement of f is not in k -DNF (and hence f is not in k -CNF) because the complement has degree- $(k + 1)$ prime implicant $\bar{x}_1\bar{x}_2 \dots \bar{x}_k\bar{x}_{k+1}$. \square

As an interesting example of decision list representation, consider the function f_n , for even n , with formula

$$f_n = x_1 \wedge (x_2 \vee (x_3 \wedge (\dots (x_{n-1} \wedge x_n) \dots))).$$

For example,

$$f_6 = x_1 \wedge (x_2 \vee (x_3 \wedge (x_4 \vee (x_5 \wedge x_6)))).$$

The function f_n is difficult to represent in DNF or CNF form: it is easily seen that both f_n and its complement have prime implicants of degree at least $n/2$, so f_n cannot be represented by a k -DNF or a k -CNF formula when $k < n/2$. However, f_n is easily represented by a 1-decision list, for

$$f_n = (\bar{x}_1, 0), (x_2, 1), (\bar{x}_3, 0), \dots, (x_{n-2}, 1), (\bar{x}_{n-1}, 0), (\bar{x}_n, 0), (\mathbf{1}, 1),$$

where we regard $\mathbf{1}$ as being represented by the empty monomial (with no literals). Note that this example almost demonstrates the strictness part of Theorem 3.2, and is, moreover, not just in k -DL, but in 1-DL.

The inclusion of k -DNF in k -DL shows that any Boolean function can be represented by a decision list in which the tests are of sufficiently high degree; that is, n -DL is the set of all Boolean functions on $\{0, 1\}^n$. So, in this sense, the decision list representation is universal. Simple counting will show that most Boolean functions need decision lists of high degree. (As we see later, using a result on polynomial threshold functions, almost every Boolean function needs tests of degree at least $k \geq \lfloor n/2 \rfloor$ in any decision list representation.)

Explicit examples can also be given of functions which have reasonably compact representations as decision lists, but which have very long DNF or CNF representations.

Let COMP_n denote the function from $\{0, 1\}^{2n} \rightarrow \{0, 1\}$ given by $\text{COMP}_n(x, y) = 1$ if and only if $\langle x \rangle > \langle y \rangle$, where, for $x \in \{0, 1\}^n$, $\langle x \rangle$ is the integer whose binary representation is x . (Thus, $\langle x \rangle = \sum_{i=1}^n 2^{n-i} x_i$.) Then, as noted in [15], for example, COMP_n can be represented by a short decision list, but has no polynomial-sized DNF or CNF formula. (It is not in the circuit complexity class AC_2^0 .) A 2-DL representation of COMP_n is

$$(\bar{x}_1 y_1, 0), (x_1 \bar{y}_1, 1), (\bar{x}_2 y_2, 0), (x_2 \bar{y}_2, 1), \dots, (\bar{x}_n y_n, 0), (x_n \bar{y}_n, 1).$$

3.2 Universality of positive-term decision lists

As we have seen, every Boolean function has a decision list representation, obtained from any DNF representation of the function. If, moreover, the function is positive and we use a positive DNF representation, then we can obtain a positive-term decision list representation. But it is fairly easy to see that any Boolean function (positive or not), can be represented by a positive-term decision list, as the following result of Guijarro *et al.* [10] establishes.

Theorem 3.3 (Guijarro *et al.* [10]) *Every Boolean function can be represented as a positive-term decision list.*

Proof: Suppose f is a given Boolean function, and construct a positive-term decision list as follows. For $y \in \{0, 1\}^n$, let T_y be the (positive) conjunction of all literals x_i which are true on y (meaning that $y_i = 1$). Then, the first-term is $(T_{11\dots 1}, f(11\dots 1))$; that is, $(x_1x_2\dots x_n, f(11\dots 1))$. The next $n-1$ terms consist (in any order) of all terms $(T_y, f(y))$ for those y having $n-1$ ones. We continue in this way, dealing with the y of weight $n-2$, and so on, until we reach $y = 00\dots 0$, so the final term of the decision list (if it is needed) is $(\mathbf{1}, f(00\dots 0))$. Clearly, f is a positive-term decision list, and it computes f .

Note that the construction in this proof will result in a very long decision list (2^n terms) of high degree (n). Some subsequent reduction in size of the list may be possible, but the question naturally arises as to how long a positive-term decision list representation of a Boolean function is compared to, say, the standard DNF and CNF representations. Guijarro *et al.* observed that there is a sequences of functions (f_n) such that the shortest length of a positive-term decision list representing f_n is exponential in the number of terms in the shortest DNF representation of f_n ; and that a corresponding result also holds for CNF representations. On the other hand, Guijarro *et al.* (invoking results of Ehrenfeucht and Haussler [6] and Fredman and Khachiyan [9]) also prove the following.

Theorem 3.4 (Guijarro *et al.* [10]) *For a Boolean function f , let $|f|_{dnf}$ and $|f|_{cnf}$ denote, respectively, the number of terms (clauses) in the shortest DNF (CNF) formulae representing f , and let $|f|$ be the larger of these two measures. Then there is a positive-term decision list representing f and having no more than $|f|^{\log^2 |f|}$ terms.*

4 Algorithmic aspects of decision lists

4.1 Membership problems

Recall that, for a class \mathcal{C} of functions, the (*functional*) *membership problem* for \mathcal{C} is as follows.

MEMBERSHIP(\mathcal{C})

Instance: A DNF formula ϕ

Question: Does the function f represented by ϕ belong to \mathcal{C} ?

A useful general result due to Hegedus and Megiddo [13] shows that MEMBERSHIP(\mathcal{C}) is NP-complete for all classes \mathcal{C} satisfying certain properties. The following definition describes these properties.

Definition 4.1 *Suppose that $\mathcal{C} = \{C_n\}$ is a class of Boolean functions. (Here, C_n maps from $\{0, 1\}^n$.) We say that a class \mathcal{C} has the projection property if*

- (i) \mathcal{C} is closed under restrictions (so, all restrictions of a function in \mathcal{C} also belong to \mathcal{C});*
- (ii) For every $n \in \mathbb{N}$, the identically-1 function $\mathbf{1}$ belongs to C_n ;*
- (iii) There exists $k \in \mathbb{N}$ such that some Boolean function on $\{0, 1\}^k$ does not belong to C_k .*

Then, we have the following result.

Theorem 4.2 (Hegedus and Megiddo [13]) *Suppose that \mathcal{C} is a class of Boolean functions having the projection property. Then MEMBERSHIP(\mathcal{C}) is NP-hard.*

For any $k < n$, the class k -DL is easily seen to have the projection property, and hence the membership problem MEMBERSHIP(k -DL) is NP-hard. (In fact, it is co-NP-complete; see [8].) (The same is true of positive-term decision lists.)

However, in the case of 1-decision lists, Eiter *et al.* [8] have established that the membership problem can be solved if the DNF is positive.

Theorem 4.3 *Deciding whether a positive DNF formula ϕ represents a function in 1-DL can be solved in polynomial time.*

4.2 Extending and learning decision lists

It is often important to determine, given a set T of points labelled ‘true’ and a set F of points labelled ‘false’, whether there is a Boolean function in a certain class \mathcal{C} that is an *extension* of the *partially defined Boolean function* $\text{pdBf}(T, F)$. In other words, the problem is to determine whether there is $f \in \mathcal{C}$ such that $f(y) = 0$ for $y \in F$ and $f(y) = 1$ for $y \in T$. In many applications, it is also important to produce such an extension, efficiently. Rivest [19] developed the following *learning algorithm*. This takes as input a sequence (or sample) $\mathbf{s} = ((y_1, b_1), \dots, (y_m, b_m))$ of labelled points of $\{0, 1\}^n$ (where $y_i \in \{0, 1\}^n$ and $b_i \in \{0, 1\}$), and finds, if one exists, a decision list in $DL(K)$ that is an extension of the sample (or, if you like, of the pdBf corresponding to the sample). (We use an ordered sample of labelled points rather than simply two subsets T, F because this is more natural in many learning contexts. However, it should be noted that the decision list output by the following algorithm does not depend on the ordering of the labelled points in the sample.)

The extension (or learning) algorithm may be described as follows. At each step in the construction of the required decision list some of the examples have been deleted, while others remain. The procedure is to run through K seeking a function $g \in K$ and a bit c such that, for all remaining points y_i , whenever $g(y_i) = 1$ then b_i is the constant Boolean value c . The pair (g, c) is then selected as the next term of the sequence defining the decision list, and all the examples satisfying g are deleted. The procedure is repeated until all the examples in \mathbf{s} have been deleted.

Let $\{g_1, g_2, \dots, g_p\}$ be an enumeration of K . The algorithm is as follows.

```

set  $I = \{1, 2, \dots, m\}$ ;  $j := 1$ ;
repeat
if for all  $i \in I$ ,  $g_j(y_i) = 1$  implies  $b_i = c$ 
    then begin select  $(g_j, c)$  ;
                delete from  $I$  all  $i$  for which  $g_j(y_i) = 1$ ;
                 $j := j + 1$  end
    else  $j := j + 1$ ;
until  $I = \emptyset$ 

```

Note, of course, that the way in which K is enumerated has an effect on the decision list output by the algorithm: different orderings of the functions in K potentially lead to different decision lists, as the following example demonstrates.

Example: Suppose we want to find a 2-decision list on five variables that is an extension of the pdBf described by the following labelled sample:

$$\begin{aligned}
\mathbf{s} &= ((y_1, b_1), (y_2, b_2), (y_3, b_3), (y_4, b_4), (y_5, b_5), (y_6, b_6)) \\
&= ((10000, 0), (01110, 0), (11000, 0), (10101, 1), (01100, 1), (10111, 1)).
\end{aligned}$$

Suppose we list the functions of $K = M_{5,2}$ in lexicographic (or dictionary) order, based on the ordering $x_1, x_2, x_3, x_4, x_5, \bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4, \bar{x}_5$ of the literals. The first few entries in the list are: the identically-1 monomial $\mathbf{1}$, x_1 , x_1x_2 , x_1x_3 . Then the algorithm operates as follows. To begin, we select the first item from the list which satisfies the required conditions. Clearly $\mathbf{1}$ will not do, because all the examples satisfy it but some have label 0 and some have label 1. Also x_1 will not do, because (for example) y_1 and y_4 both satisfy it but $b_1 \neq b_4$. However, x_1x_2 is satisfied only by y_3 , and $b_3 = 0$, so we select $(x_1x_2, 0)$ as the first term in the decision list, and delete y_3 . The subsequent steps are as follows:

- select $(x_1x_3, 1)$, delete y_4 and y_6 ;
- select $(x_1, 0)$, delete y_1 ;
- select $(x_2x_4, 0)$, delete y_2 ;
- select $(\mathbf{1}, 1)$, delete y_5 .

In this case the output decision list is therefore

$$(x_1x_2, 0), (x_1x_3, 1), (x_1, 0), (\bar{x}_1x_4, 0), (\mathbf{1}, 1).$$

Suppose instead that the functions in $K = M_{5,2}$ were enumerated instead in such a way that the smaller monomials came first; that is, we started with $\mathbf{1}$, then listed (in lexicographic order) all monomials of length 1, then all of length 2:

$$\mathbf{1}, x_1, x_2, x_3, x_4, x_5, \bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4, \bar{x}_5, x_1x_2, x_1x_3, x_1x_4, \dots$$

In this case, the decision list output by the algorithm is the 1-decision list

$$(x_5, 1), (x_1, 0), (x_4, 0), (x_2, 1).$$

This is simpler, in the sense that it is a 1-decision list rather than a 2-decision list.

It is easily verified that both decision lists are indeed extensions of the pdBf given by the sample. \square

Correctness of the algorithm in general is easily established [19, 2].

Theorem 4.4 *Suppose that K is a set of Boolean functions containing the identically-1 function, $\mathbf{1}$. Suppose that \mathbf{s} is a sample of labelled elements of $\{0, 1\}^n$. If there is an extension in $DL(K)$ of the partially defined Boolean function described by \mathbf{s} , then the above algorithm will produce such an extension.*

The extension algorithm is also efficient: when $K = M_{n,k}$, so that the class $DL(K)$ is k -DL, then the algorithm is easily seen to have running time $O(mn^{k+1})$ for fixed k . There is no guarantee, however, that the algorithm will necessarily produce a decision list that is nearly as short as it could be, as Hancock *et al.* [12] have shown.

Eiter *et al.* [8] considered 1-decision lists in some detail and were able to find an improved (that is, faster) extension algorithm. In fact, rather than a running-time of $O(mn^2)$, their algorithm has linear running time $O(mn)$. They also develop a *polynomial delay algorithm* for generating *all* 1-decision list extensions of a pdBf (when such extensions exist). Such an algorithm outputs, one-by-one, and without repetition, all extensions of the pdBf in such a way that the running time between outputs is polynomial in nm . (This is a reasonable requirement: to ask for the *total* time to generate all extensions to be polynomial in mn would be inappropriate since the number of extensions may well be exponential in mn .)

4.3 Algorithmic issues for positive-term decision lists

For positive-term decision lists, Guijarro *et al.* [10] have shown that a number of problems which are intractable for general decision lists become efficiently solvable.

The following result is useful. It shows that the question of whether two positive-term decision lists are equivalent (that is, represent the same function) can be resolved quite simply.

Theorem 4.5 (Guijarro *et al.* [10]) *There is an algorithm with running time $O(n(p+q)pq)$ which, given two positive-term decision lists on n variables, involving p and q tests, decides whether or not the decision lists represent the same function.*

Proof: Suppose the decision lists are L_1, L_2 . For any test T from L_1 and any test S from L_2 , let $y(T, S) \in \{0, 1\}^n$ have ones in precisely those positions i for which x_i appears in T or S . Let f_1, f_2 be the functions computed by L_1 and L_2 . Suppose $f_1 \neq f_2$ and let z be such that $f_1(z) \neq f_2(z)$. Let (T, c) be the first term of L_1 ‘activated’ by z (meaning that T is the first test in L_1 passed by z), The first term of L_2 activated by z is then necessarily of the form $(S, 1 - c)$. Then, as can easily be seen, $f_1(y(T, S)) = c$ and $f_2(y(T, S)) = 1 - c$. Thus there exists tests T and S of L_1, L_2 , respectively, such that $f_1(y(T, S)) \neq f_2(y(T, S))$. Conversely, of course, if such T, S exist then $f_1 \neq f_2$. So it suffices to check, for each of the pq pairs (T, S) , whether $f_1(y(T, S)) = f_2(y(T, S))$ and, for each pair, this can be done in $O(n(p+q))$ time. \square

As Guijarro *et al.* observe, the existence of efficient algorithms for other problems follows from this result. For example, to check whether a term is redundant (unnecessary), simply remove it and check the equivalence of the new decision list with the original. Furthermore, to check whether a positive-term decision list represents a positive function, one can remove from the list all redundant terms (using the redundancy-checking method just described) and check whether the remaining terms all have label 1; they do so if and only if the function is positive.

5 Properties of 1-decision lists

5.1 Threshold functions and 1-decision lists

Recall that a Boolean function t defined on $\{0, 1\}^n$ is a *threshold function* if there are $w \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ such that

$$t(x) = \begin{cases} 1 & \text{if } \langle w, x \rangle \geq \theta \\ 0 & \text{if } \langle w, x \rangle < \theta, \end{cases}$$

where $\langle w, x \rangle = w^T x$ is the standard inner product of w and x . Given such w and θ , we say that t is represented by $[w, \theta]$ and we write $t \leftarrow [w, \theta]$. The vector w is known as the *weight-vector*, and θ is known as the *threshold*. We denote the class of threshold functions on $\{0, 1\}^n$ by T_n . Note that any $t \in T_n$ will satisfy $t \leftarrow [w, \theta]$ for ranges of w and θ .

We have the following connection between 1-decision lists and threshold functions [7] (see also [3]).

Theorem 5.1 *Any 1-decision list is a threshold function.*

Proof: We prove this by induction on the number of terms in the decision list. Note that the identically-one function $\mathbf{1}$ is regarded as a monomial of length 0. Suppose, for the base case of the induction, that a decision list has just one term, and is of the form $(x_i, 1)$, or $(\bar{x}_i, 1)$, or $(\mathbf{1}, 1)$, where $\mathbf{1}$ is the identically-1 function. (Note that if it were of the form $(x_i, 0)$, $(\bar{x}_i, 0)$, or $(\mathbf{1}, 0)$ then, since a decision list outputs 0 by default, the term is redundant, and the decision list computes the identically-0 function, which is certainly a threshold function.) In the first case, the function may be represented as a threshold function by taking the weight-vector to be $(0, \dots, 0, 2, 0, \dots, 0)$, where the non-zero entry is in position i , and by taking the threshold to be 1. In the second case, we may take weight-vector $(0, \dots, 0, -2, 0, \dots, 0)$ and threshold -1 . In the third case, the function is the identically-1 function, and we may take as weight-vector the all-0 vector, and threshold 0. Assume, as the inductive hypothesis, that any decision list of length r is a threshold function, and suppose we have a decision list of length $r + 1$,

$$f = (\ell_1, c_1), (\ell_2, c_2), \dots, (\ell_{r+1}, c_{r+1}),$$

where each ℓ_i is a literal, possibly negated. We shall assume, without any loss of generality (for one can simply rename the variables or, equivalently, permute the entries of the weight vector), that $\ell_1 = x_1$ or \bar{x}_1 . By the induction hypothesis, the decision list

$$(\ell_2, c_2), \dots, (\ell_{r+1}, c_{r+1})$$

is a threshold function. Suppose it is represented by weight-vector $w = (w_1, \dots, w_n)$ and threshold θ , and let $\|w\|_1 = \sum_{i=1}^n |w_i|$ be the 1-norm of w . There are four possibilities for (ℓ_1, c_1) , as follows:

$$(x_1, 1), (x_1, 0), (\bar{x}_1, 1), (\bar{x}_1, 0).$$

Denoting by e_1 the vector $(1, 0, \dots, 0)$, and letting $M = \|w\|_1 + |\theta| + 1$, we claim that the decision list f is a threshold function represented by the weight-vector w' and threshold θ' , where, respectively,

$$\begin{aligned} w' &= w + Me_1, & \theta' &= \theta, \\ w' &= w - Me_1, & \theta' &= \theta, \\ w' &= w - Me_1, & \theta' &= \theta - M, \\ w' &= w + Me_1, & \theta' &= \theta + M. \end{aligned}$$

This claim is easy to verify in each case. Consider, for example, the third case. For $x \in \{0, 1\}^n$,

$$\langle w', x \rangle = \langle w - Me_1, x \rangle = \langle w, x \rangle - Mx_1,$$

and therefore $\langle w', x \rangle \geq \theta' = \theta - M$ if and only if

$$\langle w, x \rangle - Mx_1 \geq \theta - M.$$

If $x_1 = 0$ (in which case the decision list outputs 1), this inequality becomes $\langle w, x \rangle \geq \theta - M$. Now, for any $x \in \{0, 1\}^n$, $-\|w\|_1 \leq \langle w, x \rangle \leq \|w\|_1$, and

$$\theta - M = \theta - (\|w\|_1 + |\theta| + 1) = -\|w\|_1 - 1 + (\theta - |\theta|) \leq -\|w\|_1 - 1 < -\|w\|_1,$$

so in this case the inequality is certainly satisfied, and the output of the threshold function is 1, equal to the output of the decision list. Now suppose that $x_1 = 1$. Then the inequality

$$\langle w, x \rangle - Mx_1 \geq \theta - M$$

becomes $\langle w, x \rangle - M \geq \theta - M$, which is $\langle w, x \rangle \geq \theta$. But, by the inductive assumption, the decision list

$$f' = (\ell_2, c_2), \dots, (\ell_{r+1}, c_{r+1})$$

is a threshold function represented by the weight-vector w and threshold θ . So in this case, the output of the threshold function is 1 if and only if the output of decision list f' is 1, which is exactly how f calculates its output in this case. So we see that this representation is indeed correct. The other cases can be verified similarly.

5.2 Characterizations of 1-decision lists

Eiter *et al.* [8] obtain some results relating the class of 1-decision lists closely to other classes of Boolean function. To describe their results, we need to recall a few more definitions concerning Boolean functions.

For i between 1 and n , let $e_i \in \{0, 1\}^n$ have i th entry 1 and all other entries 0. Then, recall that a Boolean function is said to be *2-monotonic* if for each pair i and j between 1 and n , *either*

$$\text{for all } x \text{ with } x_i = 0 \text{ and } x_j = 1, f(x) \leq f(x + e_i - e_j),$$

or

$$\text{for all } x \text{ with } x_i = 0 \text{ and } x_j = 1, f(x) \geq f(x + e_i - e_j).$$

It is easily seen that threshold functions are 2-monotonic. (However, there are 2-monotonic functions that are not threshold functions.)

A Boolean function is *read-once* if there is a Boolean formula representing f in which each variable appears at most once. (So, for each j , the formula contains either x_j at most once, or \bar{x}_j at most once, but not both.)

A DNF formula is *Horn* if each term contains at most one negated literal and a function is said to be Horn if it has a representation as a Horn DNF. Given a class H of Boolean functions, the *renaming closure* of H is the set of all Boolean functions obtained from H by replacing every occurrence of some literals by their complements. (For example, we might replace every x_1 by \bar{x}_1 and every \bar{x}_1 by x_1 .)

Eiter *et al.* [8] obtained the following characterisations (among others):

Theorem 5.2 *The class 1-DL coincides with the following classes:*

- the class of all 2-monotonic read-once functions;
- the class of all read-once threshold functions;
- the renaming closure of the class of functions f such that f and \bar{f} are Horn.

In particular, while Theorem 5.1 establishes that 1-decision lists are threshold functions, the theorem just given states that the 1-decision lists are precisely those threshold functions that are also read-once.

Eiter *et al.* show that these characterisations do not extend to k -DL for $k > 1$. They do, however, have one characterisation of k -DL. Given a class H of Boolean functions, the class of *nested differences* of H is defined as the set of all Boolean functions of the form

$$h_1 \setminus (h_2 \setminus (\dots (h_{l-1} \setminus h_l)))$$

for $h_1, h_2, \dots, h_l \in H$. Eiter *et al.* prove that k -DL coincides with the nested differences of clauses with at most k -literals. (They show too that k -DL is the set of nested differences of k -CNF.)

For $k > 1$, k -decision lists are not necessarily threshold functions, but they can be described in terms of *polynomial threshold functions*, a convenient generalization of threshold functions. Theorem 5.1 shows that 1-decision lists are threshold functions. An analogous argument establishes the following.

Theorem 5.3 *Any k -decision list is a polynomial threshold function of degree k .*

It was noted in [1] that almost every Boolean function has threshold order at least $\lfloor n/2 \rfloor$. This, together with Theorem 5.3, means that almost every Boolean function will need $k \geq \lfloor n/2 \rfloor$ in order to be in k -DL.

6 Threshold decision lists

6.1 Definition and geometrical interpretation

We now consider the class of decision lists in which the individual tests are themselves threshold functions. We shall call such decision lists *threshold decision lists*, but they have also been called *neural* decision lists [16] and *linear* decision lists [21].

Suppose we are given the points in $\{0, 1\}^n$, each one labelled by the value of a Boolean function f . Of course, since there are very few threshold functions, it is unlikely that the true and false points of f can be separated by a hyperplane. One possible alternative is to try a separator of higher degree, representing f as a polynomial threshold function. Here we consider a different approach, in which we successively ‘chop off’ like points (that is, points all possessing the same label). We first use a hyperplane to separate off a set of points all having the same classification (either all are true points or all are false points). These points are then removed from consideration and the procedure is iterated until no points remain. For simplicity, we assume that at each stage, no data point lies on the hyperplane. This procedure is similar in nature to one of Jeroslow [14], but at each stage in his procedure, only positive examples may be ‘chopped off’ (not positive *or* negative). We give one example for illustration.

Example: Suppose the function f is the parity function, so that the true points are precisely those with an odd number of ones. We first find a hyperplane such that all points on one side of the plane are either positive or negative. It is clear that all we can do at this first stage is chop off one of the points since the nearest neighbours of any given point have the opposite classification. Let us suppose that we decide to chop off the origin. We may take as the first hyperplane the plane with equation $y_1 + y_2 + \dots + y_n = 1/2$. (Of course, there are infinitely many other choices of hyperplane which would achieve the same effect with respect to the data points.) We then ignore the origin and consider the remaining points. We can next chop off all neighbours of the origin, all the points which have precisely one entry equal to 1. All of these are positive points and the hyperplane $y_1 + y_2 + \dots + y_n = 3/2$ will separate them from the other points. These points are then deleted from consideration. We may continue in this manner. The procedure iterates n times, and at stage i in the procedure we ‘chop off’ all data points having precisely $(i - 1)$ ones, by using the hyperplane $y_1 + y_2 + \dots + y_n = i - 1/2$, for example. (These hyperplanes

are in fact all parallel, but this is not necessary.) □

Note that, by contrast, Jeroslow’s method [14] (described above) requires 2^{n-1} iterations in this example, since at each stage it can only ‘chop off’ one positive point.

We may regard the chopping procedure as deriving a representation of the function by a threshold decision list. If, at stage i of the procedure, the hyperplane with equation $\sum_{i=1}^n w_i y_i = \theta$ chops off true (false) points, and these lie on side of the hyperplane with equation $\sum_{i=1}^n w_i y_i > \theta$, then we take as the i th term of the threshold decision list the pair $(f_i, 1)$ (respectively, $(f_i, 0)$), where $f_i \leftarrow [w, \theta]$; otherwise we take the i th term to be $(g_i, 1)$ (respectively, $(g_i, 0)$), where $g_i \leftarrow [-w, -\theta]$.

If one applies this construction to the series of hyperplanes resulting from the Jeroslow method, a restricted form of decision list results—one in which all terms are of the form $(f_i, 1)$. But, as we saw earlier, such a decision list is quite simply the *disjunction* $f_1 \vee f_2 \vee \dots$. The problem of decomposing a function into the disjunction of threshold functions has been considered also by Hammer *et al.* [11] and Zuev and Lipkin [22]. Hammer *et al.* defined the *threshold number* of a Boolean function to be the minimum s such that f is a disjunction of s threshold functions, and they showed that there is a positive function with threshold number $\binom{n}{n/2}/n$. Zuev and Lipkin [22] showed that almost all positive functions have threshold number of this order, and that almost all Boolean functions have a threshold number that is $\Omega(2^n/n)$ and $O(2^n \ln n/n)$.

The decision lists arising from the chopping procedure are more general than disjunctions of threshold functions, just as k -decision lists are more general than k -DNF. Such threshold decision lists may provide a more compact representation of the function. (That is, since fewer hyperplanes might be used, the decision list could be smaller.)

6.2 Algorithmics and heuristics of the chopping procedure

The chopping procedure described above was in some ways merely a device to help us see that threshold decision lists have a fairly natural geometric interpretation. Furthermore, since all points of $\{0, 1\}^n$ are labelled, it is clear that the method, if implemented, would generally be inefficient. However, if only some of the points are labelled, so that we have a partially-defined Boolean function, then the chopping procedure might constitute a

heuristic for building a threshold decision list extension of the pdBf. This was considered by Marchand and Golea [16]. (See also [17].) Marchand *et al.* derive a greedy heuristic for constructing a sequence of ‘chops’. This relies on an incremental heuristic for the NP-hard problem of finding at each stage a hyperplane that chops off as many remaining points as possible. Reports on the experimental performance of their method can be found in the papers cited.

7 Threshold network representations

We now show how we can make use of the chopping procedure to find a threshold network (the simplest type of artificial neural network) representing a given Boolean function. We use linear threshold networks having just one hidden layer. Such a network will consist of k ‘hidden nodes’, each of which computes a threshold function of the n inputs. The (binary-valued) outputs of these hidden nodes are then used as the inputs to the output node, which calculates a threshold function of these. Thus, the neural network computes a threshold function of the outputs of the k threshold functions computed by the hidden nodes. If the threshold function computed by the output node is described by weight-vector $\beta \in \mathbb{R}^k$ and threshold ϕ , and the threshold function computed by hidden node i is $f_i \leftarrow [w^{(i)}, \theta^{(i)}]$, then the threshold network as a whole computes the function $f : \mathbb{R}^n \rightarrow \{0, 1\}$ given by

$$f(y) = 1 \iff \sum_{i=1}^k \beta_i f_i(y) > \phi;$$

that is,

$$f(y_1 y_2 \dots y_n) = \text{sgn} \left(\sum_{i=1}^k \beta_i \text{sgn} \left(\sum_{j=1}^n w_j^{(i)} y_j - \theta^{(i)} \right) - \phi \right),$$

where $\text{sgn}(x) = 1$ if $x > 0$ and $\text{sgn}(x) = 0$ if $x < 0$.

It is well-known that any Boolean function can be represented by a linear threshold network with one hidden layer (albeit with potentially a large number of nodes in the hidden layer). The standard way of doing so is based on the function’s disjunctive normal form.

The threshold decision list representation of a function gives rise to a different method of

representing Boolean functions by threshold networks.

We have seen that a 1-decision list is a threshold function and that a k -decision list is a polynomial threshold function of degree k . In an easy analogue of this, we see that any threshold decision list is a threshold function of threshold functions. But a threshold function of threshold functions is nothing more than a two-layer threshold network of the type considered here. So, by representing a function by a threshold decision list and then representing this as a threshold function over the threshold functions in the decision list, we obtain another method for finding a threshold network representation of a Boolean function. It is clear that the resulting representation is in general different from the standard DNF-based one. For example, the standard representation of the parity function on $\{0, 1\}^n$ will require a neural network with 2^{n-1} hidden units, whereas the representation derived from the procedure described here will require only n hidden units.

Marchand *et al.* [16] drew attention to (essentially) this link between threshold decision lists and threshold networks. (Their networks were, however, slightly different, in that they had connections between nodes in the hidden layer.)

8 Representational power of threshold decision lists

8.1 A function with long threshold decision list representation

Turan and Vatan gave a specific example of a function with a necessarily long threshold decision list representation. The inner-product modulo 2 function $\text{IP}_2 : \{0, 1\}^{2n} \rightarrow \{0, 1\}$ is given by $\text{IP}_2(x, y) = \bigoplus_{i=1}^n x_i y_i$, for $x, y \in \{0, 1\}^{2n}$, where \bigoplus denotes addition modulo 2. Turan and Vatan proved the following.

Theorem 8.1 *In any threshold decision list representation of IP_2 , the number s of terms satisfies $s \geq 2^{n/2} - 1$.*

8.2 Multi-level threshold functions

We saw in the earlier example that the parity function can be represented by a threshold decision list with n terms. We also noted that the hyperplanes in that example were parallel. By demanding that the hyperplanes are parallel, we obtain a special subclass of threshold decision lists, known as the *multi-level threshold functions*. These have been considered in a number of papers, such as [4, 18, 20], for instance.

We define the class of s -level threshold functions to be the set of Boolean functions representable by a threshold decision list of length at most s and having the test hyperplanes parallel to each other.

Geometrically, a Boolean function is an s -level threshold function if there are s parallel hyperplanes with the property that the $s + 1$ regions defined by these hyperplanes each contains only true points or only false points. Equivalently (following Bohossian and Bruck [4]), f is an s -level threshold function if there is a weight-vector $w = (w_1, w_2, \dots, w_n)$ such that

$$f(x) = F \left(\sum_{i=1}^n w_i x_i \right),$$

where the function $F : \mathbb{R} \rightarrow \{0, 1\}$ is piecewise constant with at most $s + 1$ pieces.

Bohossian and Bruck observed that any Boolean function is a 2^n -level threshold function, an appropriate weight-vector being $w = (2^{n-1}, 2^{n-2}, \dots, 2, 1)$. For that reason, they paid particular attention to the question of whether a function can be computed by a multi-level threshold function where the number of levels is polynomial. A related question considered by Bohossian and Bruck is whether a function can be computed by such a function, with polynomial weights (in addition to the restriction that the number of levels be polynomial).

It was explained earlier that, through the chopping procedure, a threshold decision list and, subsequently, a threshold network, could be produced representing a given Boolean function. The translation from threshold decision list to threshold network is established by an analogue of Theorem 5.1. From the proof of that theorem, it emerges that the weights in the resulting threshold network are, necessarily, exponential in size. It is often useful to focus on networks of the same structure, which is to say, having one ‘hidden’ layer, but which are restricted to have integer weights polynomial in n . (Any such network

can, insofar as it is regarded as computing a Boolean function, be assumed to have integer weights: we can simply scale up rational weights appropriately; and there is never a need for irrational weights since the domain is discrete.) The class of functions that can be computed by threshold networks with one hidden layer (that is, of depth 2) is denoted LT_2 , and the subset of those in which the (integer) weights can be polynomial in the number of inputs (or variables), n , is denoted $\hat{L}T_2$. Let LTM denote the set of Boolean functions f (or, more precisely, the set of sequences of Boolean functions (f_n) where f maps from $\{0, 1\}^n$) that can be computed by a multi-level threshold function with a polynomial number of levels. Then the following inclusion is valid.

Theorem 8.2 (Bohossian and Bruck [4]) *Let LTM denote the set of Boolean functions realisable by multi-level threshold functions with a polynomial number of levels. Then $LTM \subseteq \hat{L}T_2$.*

Bohossian and Bruck also obtain ‘separation’ results, which show that there are functions in $\hat{L}T_2$ but not in LTM; and that there are functions in LTM, but which are not representable with polynomial-sized weights.

9 Conclusions

In this chapter, we have looked at decision lists, a powerful and versatile way of representing Boolean functions. Decision lists have a number of interesting properties. There are, moreover, efficient algorithms for certain problems associated with classes of decision list. Allowing decision lists to be based on threshold functions allows greater generality, and draws connections with threshold networks. There are still many open problems concerning decision lists, particularly, threshold decision lists.

Acknowledgements

I am grateful to a referee for several useful comments. This work is supported in part by the IST Programme of the European Community, under the PASCAL Network of

Excellence, IST-2002-506778.

References

- [1] Martin Anthony. Classification by polynomial surfaces. *Discrete Applied Mathematics*, 61 (1995): 91–103.
- [2] Martin Anthony and Norman L. Biggs. *Computational Learning Theory: An Introduction*, Cambridge University Press, Cambridge, UK, 1992.
- [3] Martin Anthony, Graham Brightwell and John Shawe-Taylor. On specifying Boolean functions by labelled examples. *Discrete Applied Mathematics*, 61, 1995: 1–25.
- [4] V. Bohossian and J. Bruck. Multiple threshold neural logic. In *Advances in Neural Information Processing, Volume 10: NIPS'1997*, Michael Jordan, Michael Kearns, Sara Solla (eds), MIT Press, 1998.
- [5] Nader Bshouty. A subexponential exact learning algorithm for DNF using equivalence queries. *Information Processing Letters* 59(1), 1996: 37-39.
- [6] Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82, 1989: 231–246.
- [7] Andrzej Ehrenfeucht, David Haussler, Michael Kearns, and Leslie Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82, 1989: 247–261.
- [8] Thomas Eiter, Toshihide Ibaraki and Kazuhisa Makino. Decision lists and related Boolean functions. *Theoretical Computer Science*, 270(1-2), 2002: 493-524.
- [9] M. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal form. *Journal of Algorithms*, 21, 1996: 618–628.
- [10] David Guijarro, Victor Lavín and Vijay Raghavan. Monotone term decision lists. *Theoretical Computer Science*, 256, 2001: 549–575.
- [11] P. L Hammer, T. Ibaraki and U. N. Peled. Threshold numbers and threshold completions. *Annals of Discrete Mathematics* 11, 1981: 125–145.

- [12] T. Hancock, T. Jiang, M. Li and J. Tromp. Lower bounds on learning decision lists and trees. *Information and Computation*, 126(2), 1996: 114–122.
- [13] T. Hegedűs and N. Megiddo. On the geometric separability of Boolean functions. *Discrete Applied Mathematics*, 66, 1996: 205–218.
- [14] R.G. Jeroslow. On defining sets of vertices of the hypercube by linear inequalities. *Discrete Mathematics*, 11, 1975: 119–124.
- [15] Matthias Krause. On the Computational Power of Boolean Decision Lists. In Proceedings of the 19th Annual Symposium of Theoretical Aspects of Computer Science (STACS), 2002, Springer Lecture Notes in Computer Science, LNCS 2285 372–383, Springer, New York.
- [16] Mario Marchand and Mostefa Golea. On learning simple neural concepts: from halfspace intersections to neural decision lists. *Network: Computation in Neural Systems*, 4, 1993: 67–85.
- [17] M. Marchand, M. Golea and P. Ruján. A convergence theorem for sequential learning in two-layer perceptrons. *Europhys. Lett.* 11, 1990, 487.
- [18] S. Olafsson and Y. S. Abu-Mostafa. The capacity of multilevel threshold functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10 (2), 1988: 277–281.
- [19] Ronald R. Rivest. Learning Decision Lists. *Machine Learning* 2 (3), 1987: 229–246.
- [20] R. Takiyama. The separating capacity of a multi-threshold element. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7, 1985: 112–116.
- [21] György Turán and Farrokh Vatan. Linear decision lists and partitioning algorithms for the construction of neural networks. Foundations of Computational Mathematics: selected papers of a conference held at Rio de Janeiro, Springer 1997, pp 414-423
- [22] A. Zuev and L. I. Lipkin. Estimating the efficiency of threshold representations of Boolean functions. *Cybernetics* 24, 1988: 713–723. (Translated from Kibernetika (Kiev), 6, 1988: 29–37.)