Available online at www.sciencedirect.com

ScienceDirect

Discrete Applied Mathematics 156 (2008) 2459–2468

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

Counting and enumerating aggregate classifiers

Jan Adem[a,*], Yves Crama[b], Willy Gochet[c], Frits C.R. Spieksma[c]

[a] *Financial Markets, ING, Avenue Marnix 24, B-1000 Brussels, Belgium*
[b] *HEC Management School, University of Liège, Boulevard du Rectorat 7 (B31), 4000 Liège, Belgium*
[c] *Operations Research Group, FETEW, Katholieke Universiteit Leuven, Naamsestraat 69, B-3000, Leuven, Belgium*

Received 21 June 2006; received in revised form 12 September 2007; accepted 9 January 2008
Available online 5 March 2008

Abstract

We propose a generic model for the "weighted voting" aggregation step performed by several methods in supervised classification. Further, we construct an algorithm to enumerate the number of distinct aggregate classifiers that arise in this model. When there are only two classes in the classification problem, we show that a class of functions that arises from aggregate classifiers coincides with the class of self-dual positive threshold Boolean functions.
© 2008 Elsevier B.V. All rights reserved.

Keywords: Supervised classification; Weighted voting; Boolean functions; Integer sequence

1. Introduction

The supervised classification problem can be described as follows. Given is a domain $D \subseteq \mathbb{R}^P$ and a design data set $\{(\mathbf{x}_1, c_1), \ldots, (\mathbf{x}_N, c_N)\}$, where $\mathbf{x}_n \in D$ is a P-dimensional row vector of measurements describing observation n, and $c_n \in \{1, \ldots, C\}$ denotes the class containing observation n, $n \in \{1, \ldots, N\}$. The supervised classification problem consists in specifying a function $g : D \to \{1, \ldots, C\} : \mathbf{x} \mapsto g(\mathbf{x})$ that classifies any point in the domain D into one of the C classes. The function g is called a *classifier*. For an observation $(\mathbf{x}_n, c_n)$ in the data set, if $g(\mathbf{x}_n) = c_n$, then point n is correctly classified by the classifier g, otherwise it is misclassified. As c_n is given for all $n \in \{1, \ldots, N\}$, the problem is called *supervised* classification problem, in contrast with the unsupervised classification problem where the class membership of the observations in the design data set is not known beforehand [17].

In practice, there exist many methods to design classifiers [17,29]. This variety of methods, and the corresponding abundance of classifiers, gives rise to a natural question: is it possible to aggregate classifiers in such a way that the resulting aggregate classifier has more desirable characteristics than any of the original classifiers separately? In this paper, we concentrate on the following aggregation framework.

Given are a finite number of classifiers $g_1, \ldots, g_L$, $L \in \{2, 3, \ldots\}$; these L classifiers will be referred to as the *component classifiers*. For every point $\mathbf{x} \in D$, the functional value $g_l(\mathbf{x})$ can be determined for $l \in \{1, \ldots, L\}$. Now, let us assume that a nonnegative real weight α_l is associated to each component classifier g_l, $l \in \{1, \ldots, L\}$. (We

* Corresponding author. Tel.: +32 2 557 13 12; fax: +32 2 557 10 26.
E-mail address: jan.adem@ing.be (J. Adem).

0166-218X/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2008.01.017

assume that at least one of the weights α_l is strictly positive.) Then, the corresponding *aggregate classifier* g assigns each point $\mathbf{x} \in D$ to the class c^* defined by the rule:

$$g(\mathbf{x}) = c^* \quad \text{if and only if}$$
$$\sum_{l=1}^{L} \{\alpha_l \mid g_l(\mathbf{x}) = c^*\} > \sum_{l=1}^{L} \{\alpha_l \mid g_l(\mathbf{x}) = c\} \quad \text{for all } c \in \{1, \dots, C\} \setminus c^*. \tag{1}$$

Informally, the weight α_l can be seen as the voting weight associated with component classifier g_l, and the aggregation operation is a weighted majority vote based on these L values. (We will make sure below that the class c^* is uniquely defined by (1).)

Example 1. Assume that $L = 5$, $C = 4$, and that $(g_1(\mathbf{x}), g_2(\mathbf{x}), g_3(\mathbf{x}), g_4(\mathbf{x}), g_5(\mathbf{x})) = (1, 2, 1, 3, 3)$. If $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5) = (1, 1, 2, 0, 0.4)$, then $c^* = \arg\max_c\{3, 1, 0.4, 0\}$ and the aggregate classifier assigns point $\mathbf{x}$ to class 1. ■

Weighted majority voting has been used for a long time in fields such as game theory and distributed computing systems [14]. In supervised classification, aggregation procedures based on weighted majority voting have become a popular research issue in the last decade, and they now constitute an active, fast-growing field of investigation [10,15,16,28]. This interest is largely due to the promising empirical results delivered by two powerful aggregation techniques: bagging [11] and boosting [26].

Bagging relies on simple majority voting to aggregate the component classifiers, i.e., each component is given the same weight $\alpha_l = \frac{1}{L}$ for $l \in \{1, \dots, L\}$. A drawback of bagging is that the weights are fixed independently of the design data set, so that a bad component classifier is given the same weight as a good one.

Boosting takes the design data set indirectly into account by letting the weight α_l depend on e_l, where e_l is the fraction of design data set observations that are misclassified by component classifier g_l. In Ada-Boosting [15], the weight of the lth component classifier is chosen as $\alpha_l = \log((1 - e_l)/e_l)$. (Notice that in this method, the weights α_l can be negative and it is not required that at least one $\alpha_l > 0, l \in \{1, \dots, L\}$.)

Similarly, in Logical Analysis of Data (LAD, see e.g. [9]), the construction of the so-called *discriminant classifier* is based on weighted aggregation of elementary classifiers. The paper [9] proposes several ways to choose the aggregation weights: uniformly as in bagging, as a function of the accuracy of the classifier on the data set, etc.

An alternative to the above approaches is to use mathematical programming techniques to determine the weights α_l according to some criterion function defined directly on the available design data set [2]. For instance, a natural objective function would be to minimize the number of misclassifications on the design data set. It can be shown that this optimization problem is NP-hard (see [1] and the references contained therein).

In this paper, we concentrate on a question motivated by such weighted majority aggregation methods, namely on the following counting problem:

> Given L component classifiers $g_1, \dots, g_L$ taking their values in $\{1, \dots, C\}$, how many different classifiers can be defined by the aggregation rule (1)?

Notice that, when the answer to this question is not too large, and when assuming the availability of not only the number of different aggregate classifiers, but also corresponding different solutions, an "optimal" aggregate classifier can be found by complete enumeration over all possible distinct aggregate classifiers for a broad class of criteria. One obvious choice, as mentioned earlier, would be to select the aggregate classifier that yields the minimum number of misclassifications. Observe that this number may be strictly less than the number of misclassifications produced by each component classifier. Alternatively, we may also want to identify the aggregate classifier that minimizes the number of observations wrongly placed in some specific category (for instance, in a medical application, the number of patients mistakenly classified as "healthy"). We also note that other optimality criteria have been used in closely related contexts, e.g., in the determination of voting weights realizing mutual exclusion in distributed computing systems (see e.g. [4,6]).

Since the answer to the counting problem defined above may depend on the choice of the component classifiers $g_1, \dots, g_L$, we shall actually restrict our attention to a more general version of the problem, denoted **#AP**, which only depends on L and C but not explicitly on $g_1, \dots, g_L$.

The paper is organized as follows. In Section 2, the aggregation problem, called **AP**, is stated, and a framework is proposed to count the number of solutions of **AP**. Section 3 proposes an enumeration algorithm and reports on the number of nonequivalent solutions of **AP** for small values of L and C. In Section 4, we show that the class of L-bit boolean functions arising from aggregate classifiers when $C = 2$ coincides with the class of self-dual positive threshold L-bit boolean functions. The last section presents some conclusions and directions for future work.

2. Solution space of the aggregation problem

In order to describe more formally the aggregation problem, let us first introduce some definitions.

Definition 1. For L in $\{2, 3, \ldots\}$ and C in $\{2, 3, \ldots\}$, an L-tuple $(\alpha_1, \ldots, \alpha_L) \in \mathbb{R}^L$ is a *solution* of the aggregation problem **AP** if and only if

(i) $\alpha_l \geq 0$ for $l \in \{1, \ldots, L\}$ and
(ii) for every possible partition of the set $\{1, \ldots, L\}$ into at least 2 and at most C nonempty subsets, there exists a subset $\mathcal{G}^*$ in the partition such that $\sum_{l\in\mathcal{G}^*} \alpha_l > \sum_{l\in\mathcal{G}} \alpha_l$ for all other subsets $\mathcal{G}$ in the partition.

The *aggregating function* associated with the solution $(\alpha_1, \ldots, \alpha_L) \in \mathbb{R}^L_+$ is the function $f : \{1, \ldots, C\}^L \to \{1, \ldots, C\}$ such that, for all $y_1, \ldots, y_L \in \{1, \ldots, C\}$,

$$f(y_1, \ldots, y_L) = c^* \quad \text{if and only if}$$
$$\sum_{l=1}^{L}\{\alpha_l \mid y_l = c^*\} > \sum_{l=1}^{L}\{\alpha_l \mid y_l = c\} \quad \text{for all } c \in \{1, \ldots, C\} \setminus c^*. \tag{2}$$

Functions defined on $\{1, \ldots, C\}^L$ and with values in $\{1, \ldots, C\}$ are sometimes called *discrete functions*, and have been investigated in e.g. Bioch [5].

Condition (i) expresses that every component classifier is taken "positively" into account in the aggregation process. This is a rather natural assumption. Together with condition (ii), it also implies that at least one component classifier must receive a strictly positive weight.

Condition (ii) ensures that the aggregate classifier given by (1), and equivalently the aggregating function given by (2), are well-defined functions. To see this, for a given point $\mathbf{x} \in D$, define the C sets $\mathcal{G}_c(\mathbf{x}) = \{l \mid g_l(\mathbf{x}) = c\}$, $c \in \{1, \ldots, C\}$. The sets $\mathcal{G}_c(\mathbf{x})$ induce a partition of the set $\{1, \ldots, L\}$ into at most C nonempty subsets. Conversely, every partition of the set $\{1, \ldots, L\}$ into at most C nonempty subsets represents a way in which the classifications of the component classifiers can potentially differ; that is, every such partition might arise from the collection $\{\mathcal{G}_1(\mathbf{x}), \ldots, \mathcal{G}_C(\mathbf{x})\}$ for some observation $\mathbf{x}$.

Thus, condition (ii) imposes that there is always a unique aggregate classification for all possible classifications assigned by the individual component classifiers. Notice that in the statement of the condition, it is not relevant which value from $\{1, \ldots, C\}$ is associated to which nonempty subset in the partition. What matters in the analysis of the solution space is the mechanism of aggregation, not its outcome.

The counting problem associated with the aggregation problem **AP** can now be more formally stated:

> [**#AP**] For all L in $\{2, 3, \ldots\}$ and for all C in $\{2, 3, \ldots\}$, compute the number of aggregating functions of the form (2).

In order to structure the solution space of **AP** further, let $\mathcal{T}$ be the set of all the possible partitions of the set $\{1, \ldots, L\}$ into at least 2, and at most C nonempty subsets. Now, order the elements of $\mathcal{T}$ in any order and label them from 1 to $|\mathcal{T}|$. Denote by $\mathcal{P}_t$ the tth partition in $\mathcal{T}$, $t \in \{1, \ldots, |\mathcal{T}|\}$.

Let $(\alpha_1, \ldots, \alpha_L)$ be a solution of **AP** and denote by $\mathcal{G}_t^*$ the (unique) subset in $\mathcal{P}_t$ such that $\sum_{l\in\mathcal{G}_t^*} \alpha_l > \sum_{l\in\mathcal{G}} \alpha_l$ for all other subsets $\mathcal{G} \in \mathcal{P}_t$, $t \in \{1, \ldots, |\mathcal{T}|\}$. Condition (ii) in Definition 1 implies that $\mathcal{G}_t^*$ is well defined. Therefore, exactly one $|\mathcal{T}|$-dimensional vector $[\mathcal{G}_1^* \ldots \mathcal{G}_{|\mathcal{T}|}^*]$ can be associated with every solution $(\alpha_1, \ldots, \alpha_L)$. This $|\mathcal{T}|$-dimensional vector will be called the *decision vector* of the solution. It is easy to see that it completely characterizes the aggregating function f associated to $(\alpha_1, \ldots, \alpha_L)$ by the voting rule (2).

Definition 2. Let $(\alpha_1, \dots, \alpha_L)$ and $(\beta_1, \dots, \beta_L)$ be solutions of **AP**. We say that $(\alpha_1, \dots, \alpha_L)$ and $(\beta_1, \dots, \beta_L)$ are *equivalent solutions* if and only if their decision vectors are identical, i.e., if they define the same aggregating function.

The above definitions put some useful structure on the solution space of **AP**. Let $\mathcal{S}_{\text{sol}} = \{(\alpha_1, \dots, \alpha_L) \mid (\alpha_1, \dots, \alpha_L)$ is a solution of **AP**$\}$. For all values of L and C, the set $\mathcal{S}_{\text{sol}}$ contains infinitely many elements. However, the equivalence relation introduced in Definition 2 partitions the set $\mathcal{S}_{\text{sol}}$ into Q equivalence classes $\mathcal{S}^q_{\text{sol}}$ with $q \in \{1, \dots, Q\}$. Therefore, counting the number of distinct aggregate classifiers really amounts to computing the number Q of nonequivalent solutions of **AP**. Formally, we define:

Definition 3. For all L in $\{2, 3, \dots\}$ and for all C in $\{2, 3, \dots\}$, $Q(L, C)$ is the number of nonequivalent solutions of **AP**, i.e., the number of distinct aggregating functions with L component classifiers and C classes.

It is obvious that, for all possible values of L and C, $Q(L, C)$ is finite. This framework leads to a discrete, rather than continuous, representation of the solution space of **AP**. As already mentioned in the Introduction, this suggests that for certain objective functions, an "optimal" aggregate classifier could be found by complete enumeration of all $(Q(L, C))$ nonequivalent solutions of **AP**. In view of this, we now turn to two related questions.

Question 1 (*The Counting Problem*). Compute $Q(L, C)$.

Question 2 (*The Enumeration Problem*). Generate $Q(L, C)$ nonequivalent solutions of **AP**.

In the next section, these questions will be partially answered. Since we are not able to come up with an analytic or recursive expression for $Q(L, C)$ as a function of L and C (not even when $C = 2$; see our discussion in Sections 4 and 5), an algorithm to compute $Q(L, C)$ will be presented as a best alternative. This enumeration algorithm will also provide an answer to Question 2.

3. An algorithm to compute $Q(L, C)$

In Section 2, we have introduced $\mathcal{T}$, the set of all partitions of the set $\{1, \dots, L\}$ into at least 2 and at most C nonempty subsets, and we have denoted by $\mathcal{P}_t$ the tth element in $\mathcal{T}$. The cardinality of $\mathcal{T}$ is

$$|\mathcal{T}| = \sum_{k=2}^{\min\{L,C\}} S(L, k)$$

where $S(L, k)$ is the Stirling number of the second kind, representing the number of ways to partition a set of L elements into exactly k nonempty subsets [12,30].

Any $|\mathcal{T}|$-dimensional vector for which the tth entry, $t \in \{1, \dots, |\mathcal{T}|\}$, is an element of $\mathcal{P}_t$ is called a *candidate decision vector*. The number of different partition decision vectors is

$$\prod_{t=1}^{|\mathcal{T}|} |\mathcal{P}_t|, \text{ which can be rewritten as: } \prod_{k=2}^{\min\{L,C\}} k^{S(L,k)}$$

since $|\mathcal{P}_t| \in \{1, \dots, \min\{L, C\}\}$ for $t \in \{1, \dots, |\mathcal{T}|\}$.

The number of candidate decision vectors for a given value of L and C gives a (weak) upper bound for $Q(L, C)$. Observe that the parameter C only influences the upper bound through the number $\min\{L, C\}$. If $L \leq C$, the parameter C does not determine the upper bound.

By Definitions 1 and 2, in order to generate all solutions of **AP**, it "suffices" to generate all candidate decision vectors and check if there exists a solution that corresponds to each candidate. If so, the candidate decision vector is a decision vector. The test comes down to the following question:

[**FP**] Is the system of strict linear inequalities

$$\sum_{l \in \mathcal{G}^*_t} \alpha_l - \sum_{l \in \mathcal{G}} \alpha_l > 0 \quad (\mathcal{G} \in \mathcal{P}_t \setminus \{\mathcal{G}^*_t\}, t \in \{1, \dots, |\mathcal{T}|\})$$

$$\alpha_l \geq 0 \quad (l \in \{1, \dots, L\})$$

feasible?

As solutions to **AP** can be multiplied by a real number $r > 0$ without changing the corresponding aggregate classifier, **FP** is equivalent to the problem **FP**-ϵ hereunder:

[**FP**-ϵ] Let $\epsilon \in \mathbb{R}$, $\epsilon > 0$. Is the system of linear inequalities

$$\sum_{l \in \mathcal{G}_t^*} \alpha_l - \sum_{l \in \mathcal{G}} \alpha_l \geq \epsilon \quad (\mathcal{G} \in \mathcal{P}_t \setminus \{\mathcal{G}_t^*\}, t \in \{1, \ldots, |\mathcal{T}|\})$$
$$\alpha_l \geq 0 \quad (l \in \{1, \ldots, L\})$$

feasible?

Note that **FP**-ϵ can be solved by linear programming techniques. These techniques also yield a feasible solution if there exists one. Hence, in this way, they can be used to answer Questions 1 and 2 at the same time and at the same computational cost.

The approach that we have underlined is computationally expensive, however, since **FP**-ϵ needs to be solved for *all* candidate decision vectors. In order to speed it up, we observe now that every decision vector must obey the following consistency rule.

Consistency Rule. Consider an arbitrary decision vector $(\mathcal{G}_1^*, \ldots, \mathcal{G}_{|\mathcal{T}|}^*)$, and let $t \in \{1, \ldots, |\mathcal{T}|\}$. For every partition $\mathcal{P}_s$, $s \in \{1, \ldots, |\mathcal{T}|\}$, $s \neq t$, if
(i) there exists an element $\mathcal{G}_s \in \mathcal{P}_s$ such that $\mathcal{G}_t^* \subseteq \mathcal{G}_s$ and,
(ii) every element $\mathcal{G}$ of $\mathcal{P}_s$, $\mathcal{G} \neq \mathcal{G}_s$, is a subset of some element $\mathcal{G}'$ of $\mathcal{P}_t$, $\mathcal{G}' \neq \mathcal{G}_t^*$,
then it must hold that $\mathcal{G}_s$ is the sth entry in the decision vector, i.e., $\mathcal{G}_s = \mathcal{G}_s^*$.

The consistency rule is obviously valid: indeed, the total weight of $\mathcal{G}_s$ is at least as large as the weight of $\mathcal{G}_t^*$, while the weight of any other set $\mathcal{G}$ in $\mathcal{P}_s$ is at most the weight of $\mathcal{G}'$, which is smaller than the weight of $\mathcal{G}_t^*$.

As a consequence, rather than enumerating all the possible candidate decision vectors, only those will be enumerated that do not violate the consistency rule. This strongly reduces the number of feasibility problems **FP**-ϵ that need to be solved. For example, if $L = 4$ and $C = 3$ only 116 feasibility checks are required while the number of candidate decision vectors is $2^7 3^6 = 93312$.

Below, an informal description is given of an algorithm that enumerates and determines the number of nonequivalent solutions of **AP**, i.e., $Q(L, C)$, for a given value of L and C.

```
ENUMERATION ALGORITHM
 Input: A number $L \in \{2, 3, \ldots\}$, a number $C \in \{2, 3, \ldots\}$.
    1. Generate all $\mathcal{P}_t$ and label them in any order from 1 to $|\mathcal{T}|$.
       Set $q = 0$.
    2. Take the first entry in a to-be-built candidate decision vector, $t = 1$.
    3. Choose a subset $\mathcal{G}_t$ in $\mathcal{P}_t$ and put subset $\mathcal{G}_t$ at entry $t$ of the to-be-built candidate
       decision vector, $\mathcal{G}_t^* = \mathcal{G}_t$.
    4. If possible, fill in unfilled entries of the to-be-built candidate decision vector
       by application of the consistency rule.
    5. If all the entries of the to-be-built candidate decision vector are filled in, a
       candidate decision vector has been built and proceed to step 6; else update $t$ to
       the next unfilled entry in the to-be-built candidate decision vector and go to
       step 3.
    6. Use FP-$\epsilon$ to check if there exists a solution that would give the candidate decision
       vector. If so, the candidate decision vector is a decision vector, $q \leftarrow q + 1$ and the
       solution is saved.
    7. Empty the entries of the to-be-built candidate decision vector filled up in the
       last step 3 and 4.
    8. If there exists a subset $\mathcal{G}_t$ in $\mathcal{P}_t$ which has not yet been chosen, go to step 3; else
       if possible, set $t$ to its previous value and go to step 8; else go to step 9.
    9. Set $Q = q$.
Output: A number $Q$, a set of $Q$ solutions.
```

Table 1
$Q(L, C)$ for small dimensions

L	$C = 2$	$C = 3$	$C = 4$	$C = 5$	$C = 6$
2	2	2	2	2	2
3	4	6	6	6	6
4	12	76	84	84	84
5	81	7 625	13 805	14 025	14 025
6	1 684	?	?	?	?
7	122 921	?	?	?	?
8	?	?	?	?	?

In Table 1, the values for $Q(L, C)$ for small values of L and C are given. A question mark indicates that it was not possible to calculate $Q(L, C)$ within 24 hours of computation time on a PENTIUM III 550 MHz computer.

Given our computational resources, this table is currently the best answer that we can provide to Question 1. For all the cases where $Q(L, C)$ can be found, *also* Question 2 is answered as the enumeration algorithm provides a set of $Q(L, C)$ solutions, each of which is associated to a distinct aggregate classifier. This set then suffices to determine, for example, a set of weights which minimize the number of misclassified observations for any instance of the aggregation problem with $L \leq 5$ component classifiers and any number C of classes (remember that, when $C \geq L$, $Q(L, C)$ depends on L only). These exact solutions provide easy-to-obtain lower bounds for larger aggregation problems and, in this sense, might prove useful in the development of algorithms to solve larger aggregation problems. Note that, in contrast to the mathematical programming approach in [2], the influence on the computational performance of the number of observations N in the design data set is negligible.

From Table 1, it is clear that the numbers $Q(L, C)$ grow extremely fast with increasing L and, due to limited computational resources, the counting algorithm will not be able to find $Q(L, C)$ when L gets large. Interestingly, none of the sequences in the table (e.g. 2, 6, 84, 14025, ... or 2, 4, 12, 81, 1684, 122921, ...) can be directly found in the on-line encyclopedia of integer sequences provided by Sloane [27]. We will come back to this issue in the next Section.

4. Aggregate classifiers and Boolean functions

In this section we explore the connection between the aggregation problem **AP** with $C = 2$ and the enumeration of n-bit *self-dual*, *positive*, *threshold* Boolean functions. Section 4.1 recalls the necessary definitions from Boolean function theory, and in Section 4.2 we prove the equivalence between aggregating Boolean functions and Boolean functions that are self-dual, monotone, and threshold.

4.1. Boolean functions

Fundamental facts and definitions about Boolean functions can be found in [3,7,8,14,23]. Let $n \in \{1, 2, \ldots\}$.

Definition 4. An *n-bit Boolean function* is a function

$$f : \{0, 1\}^n \to \{0, 1\} : (b_1, \ldots, b_n) \mapsto f(b_1, \ldots, b_n).$$

When b is an element of $\{0, 1\}$, we let $\bar{b} = 1 - b$.

Definition 5. An n-bit Boolean function f is *self-dual* if for all $(b_1, \ldots, b_n) \in \{0, 1\}^n$, $f(b_1, \ldots, b_n) = \bar{f}(\bar{b}_1, \ldots, \bar{b}_n)$.

For any $(b_1, \ldots, b_n), (b'_1, \ldots, b'_n) \in \{0, 1\}^n$, we write $(b_1, \ldots, b_n) \leq (b'_1, \ldots, b'_n)$ if $b_j \leq b'_j$ for $j \in \{1, \ldots, n\}$.

Definition 6. An n-bit Boolean function f is *positive* (sometimes called *monotone*) if $f(b_1, \ldots, b_n) \leq f(b'_1, \ldots, b'_n)$ whenever $(b_1, \ldots, b_n) \leq (b'_1, \ldots, b'_n)$.

Definition 7. An n-bit Boolean function f is *threshold* if there exist $(n+1)$ real numbers $\alpha_1, \ldots, \alpha_n, \theta$ such that, for all $(b_1, \ldots, b_n) \in \{0,1\}^n$,

$$f(b_1, \ldots, b_n) = 1 \quad \text{iff} \ \sum_{j=1}^{n} \alpha_j b_j > \theta. \tag{3}$$

We denote the set of n-bit (respectively, self-dual, positive, threshold) Boolean functions by $\mathcal{B}$ (respectively, $\mathcal{B}_{sd}$, $\mathcal{B}_p$, $\mathcal{B}_t$). Counting the number of Boolean functions with various properties has been the subject of numerous mathematical studies. It is easy to check that the total number of n-bit Boolean functions is 2^{2^n}, and that the number of self-dual n-bit Boolean functions is $2^{2^{n-1}}$. Determining the number of positive n-bit Boolean functions is known as Dedekind's problem. Many mathematicians contributed to this problem but despite their efforts, the exact number of positive n-bit Boolean functions is known for small values of n only [22]. Kleitman [19] proved that $\log_2 |\mathcal{B}_p|$ is asymptotic to the middle binomial coefficient $\binom{n}{[n/2]}$ (see also [20,21]). Bioch and Ibaraki [6] enumerate all self-dual, positive functions for $n \leq 7$.

Counting and tabulating threshold functions has been a main topic of investigation in electrical engineering [23]. Here again, the exact value of $|\mathcal{B}_t|$ is known for small values of n only (see e.g. [6,23,24]). Asymptotically, it has been proved that $(\log_2 |\mathcal{B}_t|)/n^2$ approaches 1 as n grows large [3,31], meaning in particular that the class of threshold functions is quite small with respect to the classes of self-dual or positive functions.

4.2. The link between classifiers with $C = 2$ and Boolean functions

For classification problems with $C = 2$ classes, in a slight departure from our usual notations, let us encode the two classes by 0 and 1, respectively. Thus, in this case, aggregating functions are L-bit Boolean functions (by Definition 1), and we are going to show that they correspond exactly to self-dual, positive, threshold Boolean functions.

Proposition. *A Boolean function is an aggregating function if and only if it is self-dual, positive and threshold.*

Proof. For $C = 2$, consider the aggregating function f associated to a solution $(\alpha_1, \ldots, \alpha_L) \in \mathbb{R}_+^L$ of **AP** as in Definition 1. Condition (2) translates to: for all $b_1, \ldots, b_L \in \{0,1\}$,

$$f(b_1, \ldots, b_L) = 1 \quad \text{iff} \ \sum_{l=1}^{L} \alpha_l b_l > \sum_{l=1}^{L} \alpha_l (1 - b_l), \tag{4}$$

or equivalently

$$f(b_1, \ldots, b_L) = 1 \quad \text{iff} \ \sum_{l=1}^{L} \alpha_l b_l > \frac{1}{2} \sum_{l=1}^{L} \alpha_l. \tag{5}$$

The Boolean function f is positive (since $\alpha_j \geq 0$ for all j), threshold (by (5)), and self-dual: indeed, in view of (4),

$$f(1 - b_1, \ldots, 1 - b_L) = 1 \quad \text{iff} \ \sum_{l=1}^{L} \alpha_l (1 - b_l) > \sum_{l=1}^{L} \alpha_l b_l \quad \text{iff} \ f(b_1, \ldots, b_L) = 0, \tag{6}$$

meaning that $f(b_1, \ldots, b_L) = \bar{f}(\bar{b}_1, \ldots, \bar{b}_L)$ as required for self-duality.

Conversely, let f be a self-dual, positive, threshold L-bit Boolean function. In view of Definition 7, f is associated to a set of weights $\alpha_1, \ldots, \alpha_L$, and to a threshold value θ satisfying (3). It is well known, and easy to check, that the weights $\alpha_1, \ldots, \alpha_L$ can be chosen to be nonnegative when f is positive (see [14,23]). Then, we claim that these weights define a solution of **AP**. Indeed, as in condition (ii) of Definition 1, consider an arbitrary partition of the set $\{1, \ldots, L\}$ into $\{\mathcal{G}_0, \mathcal{G}_1\}$ (where $\mathcal{G}_1$ may be empty). For all $l \in \{1, \ldots, L\}$, write $b_l = 0$ if $l \in \mathcal{G}_0$ and $b_l = 1$ if $l \in \mathcal{G}_1$. By self-duality, $f(b_1, \ldots, b_L) = \bar{f}(\bar{b}_1, \ldots, \bar{b}_L)$, and we can assume without loss of generality that $f(b_1, \ldots, b_L) = 0$, $f(\bar{b}_1, \ldots, \bar{b}_L) = 1$ (the other case being symmetrical). Thus, by Definition 7,

$$\sum_{l=1}^{L} \alpha_l (1 - b_l) > \theta \geq \sum_{l=1}^{L} \alpha_l b_l,$$

Table 2
Table from [23,6]

L	Number of functions
2	0
3	1
4	4
5	46
6	1 322
7	112 519

hence

$$\sum_{l\in\mathcal{G}_0}\alpha_l > \sum_{l\in\mathcal{G}_1}\alpha_l.$$

This shows that the weights $\alpha_1,\ldots,\alpha_L$ satisfy condition (ii) in Definition 1, i.e. $(\alpha_1,\ldots,\alpha_L)$ is a solution of **AP** and f is an aggregating function. □

In view of this result, computing $Q(L,2)$ is tantamount to computing the number of self-dual, positive, threshold L-bit Boolean functions. The value of $Q(L,2)$ is found in the first column of Table 1 for $L \le 7$.

Interestingly, enumeration algorithms for the number of self-dual, positive, threshold L-bit Boolean functions have been previously considered in the Boolean literature. (To the best of our knowledge, all such algorithms actually rely on the solution of the corresponding complete enumeration problem.) For $L \le 7$, Muroga [23] and Bioch and Ibaraki [6] tabulate the results as shown in Table 2. (This same sequence is also erroneously listed in the On-Line Encyclopedia of Integer Sequences [27] for the number of self-dual threshold functions.)

The difference between the two sequences in Tables 1 and 2 can be explained by the fact that [23,6] count the number of self-dual positive threshold Boolean functions that effectively depend on *all their L variables*, while we actually count self-dual positive threshold functions of *at most L variables*, including those that may effectively depend on a subset of their L variables only (since some of the weights α_l may be zero in (4)). For instance, the function $f(b_1,b_2)=b_1$ is self-dual, positive and threshold, but it does not effectively depend on its second variable.

Let $S(L)$ be the Lth element of the sequence considered in [23,6]. Then, it is easy to see that $Q(L,2) = \sum_{n=1}^{L}\binom{L}{n}S(n)$ must hold for all $L\in\{2,3,\ldots\}$, and to check this relation for Tables 1 and 2.

Finally, let us observe that the results presented in previous sections usually have simple Boolean interpretations when $C=2$. For instance, the system of inequalities **FP** in Section 3 has been classically used for the recognition of threshold Boolean functions, and the 'Consistency Rule" simply amounts to ensuring that the function is positive and concentrating on "minimal true points" or "maximal false points" of the function (see e.g. [13,14,23,25] for details). Also (as pointed out by one of the referees), it is known that every self-dual Boolean function is a nested composition of the basic majority function of three variables, i.e., the threshold function $m(b_1,b_2,b_3)$ defined by $\alpha_1=\alpha_2=\alpha_3=\theta=1$; see e.g. [7,8,18]. So, when $C=2$, every aggregating function can be viewed as a composition of simple majority voting aggregators.

5. Conclusions and open questions

This paper considers a generic model for the "weighted voting" aggregation step performed by several classification methods. After having observed that the number of distinct aggregate classifiers is finite for every set of component classifiers $g_1,g_2,\ldots,g_L$, we have presented an algorithm that is able to count and to generate all nonequivalent solutions of the aggregation problem. Interestingly, the resulting sequence of numbers $Q(L,C)$ appears to be new.

This complete enumeration approach allows us to find exact solutions for certain optimization versions of the aggregation problem (e.g., for minimizing the number of misclassified observations). Evidently, this approach can only be successful for instances of the aggregation problem where L and C are small.

For $C=2$, we have also established the link between the aggregation problem and previous results concerning L-bit self-dual, positive, threshold Boolean functions. This connection shows that the class of aggregating functions appears to be an intriguing new class of discrete functions, and that our model for the aggregation problem has far-reaching ramifications. This opens up new opportunities for theoretical investigations of the aggregation problem.

In particular, it may prove interesting to investigate more thoroughly the complexity of the computational problems raised in this paper. When $C = 2$, as pointed out by one of the referees, the question: "does the vector $(\alpha_1, \dots, \alpha_L)$ define a feasible solution of the aggregation problem" is coNP-complete; indeed, checking condition (ii) in Definition 1 (i.e., checking self-duality of the corresponding aggregating function) is tantamount to solving the well-known PARTITION problem. On the other hand, analyzing the complexity of the counting problem [#AP] is a more challenging question, even when $C = 2$. The function $Q(L, C)$ only depends on two argument L, C, and it seems unlikely that the value of $Q(L, C)$ can be computed in time polynomial in the input size ($\log L + \log C$), nor perhaps even in time polynomial in $L + C$. In particular, no closed-form expression or recursive formula appears to be known in the literature for the number $Q(L, 2)$ of positive, self-dual threshold functions on L variables, or for the number of threshold functions, in spite of the fact that this number is asymptotically known ([3,31]). Also, the complexity of generating all aggregating functions, for given values of L and C, is currently unknown. In particular, the algorithm in Section 3 is not efficient, as it examines many candidate decision vectors which must eventually be rejected. In principle, it may be possible to generate efficiently all aggregating functions in an incremental fashion, for successive increasing values of L and C. At this time, however, we do not even know how to solve the problem when $C = 2$, that is, how to generate efficiently all self-dual, positive, threshold functions of L variables. (Note that efficient algorithms exist for the generation of all self-dual positive Boolean functions; see [6].)

Acknowledgements

We thank Peter Goos and the anonymous referees for their insightful comments.

References

[1] J. Adem, Mathematical programming approaches for the supervised classification problem, Ph.D. Thesis, Katholieke Universiteit Leuven, 2004.
[2] J. Adem, W. Gochet, Aggregating classifiers with mathematical programming, Computational Statistics and Data Analysis 47 (2004) 791–807.
[3] M. Anthony, Discrete mathematics of neural networks: Selected topics, in: SIAM Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, 2001.
[4] D. Barbara, H. Garcia-Molina, The reliability of voting mechanisms, IEEE Transactions on Computers C-36 (1987) 1197–1208.
[5] J.C. Bioch, Dualization, decision lists and identification of monotone discrete functions, Annals of Mathematics and Artificial Intelligence 24 (1998) 69–91.
[6] J. Bioch, T. Ibaraki, Generating and approximating nondominated coteries, IEEE Transactions on Parallel and Distributed Systems 6 (1995) 905–914.
[7] J. Bioch, T. Ibaraki, Decompositions of positive self-dual Boolean functions, Discrete Mathematics 140 (1995) 23–46.
[8] J. Bioch, T. Ibaraki, K. Makino, Minimum self-dual decompositions of positive dual-minor Boolean functions, Discrete Applied Mathematics 97 (1999) 307–326.
[9] E. Boros, P.L. Hammer, T. Ibaraki, A. Kogan, E. Mayoraz, I. Muchnik, An implementation of logical analysis of data, IEEE Transactions on Knowledge and Data Engineering 12 (2000) 292–306.
[10] P. Bühlmann, B. Yu, Analyzing bagging, The Annals of Statistics 30 (1996) 927–961.
[11] L. Breiman, Bagging predictors, Machine Learning 24 (1996) 123–140.
[12] J. Conway, R. Guy, The Book of Numbers, Springer-Verlag, New York, 1996.
[13] Y. Crama, Dualization of regular Boolean functions, Discrete Applied Mathematics 16 (1987) 79–85.
[14] Y. Crama, P.L. Hammer, Boolean functions — Theory, algorithms, and applications, Cambridge University Press (in press) http://www.rogp.hec.ulg.ac.be/crama/.
[15] Y. Freund, R. Schapire, Experiments with a new boosting algorithm, in: Proceedings of the Thirteenth International Conference on Machine Learning, Bari, Italy, 1996, pp. 148–156.
[16] J. Friedman, T. Hastie, R. Tibshirani, Additive logistic regression: A statistical view of boosting, The Annals of Statistics 28 (2000) 337–374.
[17] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer, 2001.
[18] T. Ibaraki, T. Kameda, A theory of coteries: Mutual exclusion in distributed systems, IEEE Transactions on Parallel and Distributed Systems 4 (1993) 779–794.
[19] D. Kleitman, On Dedekind's problem: The number of monotone Boolean functions, in: Proceedings of the American Mathematical Society, 21, 1969, pp. 677–682.
[20] D. Kleitman, G. Markowsky, On Dedekind's problem: The number of isotone Boolean functions. II, Transactions of the American Mathematical Society 213 (1975) 373–390.
[21] A.D. Korshunov, Families of subsets of a finite set and closed classes of Boolean functions, in: P. Frankl, et al. (Eds.), Extremal Problems for Finite Sets, János Bolyai Mathematical Society, Budapest, Hungary, 1994, pp. 375–396.
[22] Mathpages. Dedekind's Problem, published electronically at http://www.mathpages.com/home/kmath030.htm, 2003.
[23] S. Muroga, Threshold Logic and its Applications, Wiley-Interscience, New York, 1971.

[24] S. Muroga, T. Tsuboi, C. Baugh, Enumeration of threshold functions of eight variables, IEEE Transactions on Computers 19 (1970) 818–825.
[25] U.N. Peled, B. Simeone, Polynomial-time algorithms for regular set-covering and threshold synthesis, Discrete Applied Mathematics 12 (1985) 57–69.
[26] R. Schapire, The strength of weak earnability, Machine Learning 5 (1990) 197–227.
[27] N. Sloane, The On-Line Encyclopedia of Integer Sequences, published electronically at http://www.research.att.com/˜njas/sequences/, 2003.
[28] A. Tsybakov, Optimal aggregation of classifiers in statistical learning, The Annals of Statistics 32 (2004) 135–166.
[29] A. Webb, Statistical Pattern Recognition, Arnold, London, 1999.
[30] E. Weisstein, Stirling Number of the Second Kind, from MathWorld — A Wolfram Web Resource published electronically at http://mathworld.wolfram.com/StirlingNumberoftheSecondKind.html, 2004.
[31] Yu.A. Zuev, Asymptotics of the logarithm of the number of threshold functions of the algebra of logic, Soviet Mathematics Doklady 39 (1989) 512–513.