DEPARTMENT OF APPLIED ECONOMICS

RESEARCH REPORT

COUNTING AGGREGATE CLASSIFIERS

Jan Adem • Willy Gochet • Frits Spieksma

OR 0451

Counting Aggregate Classifiers

Jan Adem, Willy Gochet, Frits Spieksma

Department of Applied Economics, Katholieke Universiteit Leuven,
Naamsestraat 69, 3000 Leuven, Belgium
e-mail: `firstname.name@econ.kuleuven.be`

Abstract: There are many methods to design classifiers for the supervised classification problem. In this paper, we study the problem of aggregating classifiers. We construct an algorithm to count the number of distinct aggregate classifiers. This leads to a new way of finding a best aggregate classifier. When there are only two classes, we explore the link between aggregating classifiers and n-bit boolean functions. Further, the sequence of the number of distinct aggregated classifiers appears to be new.

Keywords: supervised classification, weighted majority vote, boolean function, integer sequence.

1 Introduction

The supervised classification problem can be described as follows. Given is a design data set $\{(\mathbf{x}_1, c_1), ..., (\mathbf{x}_N, c_N)\}$ where $\mathbf{x}_n$ is a P-dimensional row vector of measurements describing pattern n, and $c_n \in \{1, 2, ..., C\}$ denotes the class that pattern n belongs to, $1 \leq n \leq N$. The supervised classification problem consists of specifying a function $g : \mathrm{IR}^P \mapsto \{1, ..., C\} : \mathbf{x} \mapsto g(\mathbf{x})$ that classifies any pattern with unknown class membership into one of the C classes as accurately as possible. The function g is called a classifier. If $g(\mathbf{x}_n) = c_n$, then pattern n is correctly classified by the classifier g, otherwise it is misclassified. As c_n is given for $n \in \{1, ..., N\}$, the problem is called the supervised classification problem, in contrast with the unsupervised classification problem where one does not know the class membership of the design data set patterns beforehand [9].

The supervised classification problem can be very hard to solve. For instance, even when $C = 2$, finding a $(P-1)$-dimensional hyperplane minimizing the number of misclassifications on the design data set is a NP-hard problem. In fact, even the existence of an algorithm that has a constant worst-case ratio would imply P = NP [10].

There exist many methods to design classifiers [9, 18]. This variety of methods and the corresponding abundance of classifiers gives rise to an obvious question: are there ways to aggregate classifiers such that the aggregated classifier has more desirable characteristics than any of the classifiers separately?

This question is the subject of this paper; thus, instead of solving the supervised classification problem by designing a classifier, we assume that we are given a number of classifiers. More concrete, we consider the following setting. Given are a finite number of classifiers $g_1, ..., g_L$, $L \in \{1, 2, ...\}$. These L classifiers will be referred to as the *component classifiers*. For each pattern n of the design data set, the (unique) functional value $g_l(\mathbf{x}_n)$ is known, $l \in \{1, ..., L\}$.

Aggregating component classifiers to obtain an aggregate classifier can be explained as follows. A nonnegative real weight α_l is associated to each component classifier g_l, $l \in \{1, ..., L\}$. Hereby, it is assumed that at least one of the weights $\alpha_l > 0$, $l \in \{1, ..., L\}$. Let $I(\cdot)$ be the indicator function, i.e. $I(\cdot) = 1$ if the argument is true and $= 0$ otherwise. The aggregated classifier classifies a pattern n into the class c^* for which $c^* = \arg\max_c\{\sum_{l=1}^{L} \alpha_l I(g_l(\mathbf{x}_n) = c)\}$. Informally, the weight α_l can be seen as the voting power associated to component classifier g_l and the aggregation operation is a simple majority vote based on these L weights. Weighted majority vote procedures are also used in fields such as game theory and distributed computing systems [7].

> **Example 1.** Let $L = 5$, $C = 4$ and $(g_1(\mathbf{x}_n), g_2(\mathbf{x}_n), g_3(\mathbf{x}_n), g_4(\mathbf{x}_n), g_5(\mathbf{x}_n)) = (1,2,1,3,3)$. 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 aggregated classifier will classify pattern n in class 1. ■

In the last decade, aggregation of classifiers has become a popular issue in the supervised classification literature. This is largely due to the promising empirical results of two powerful aggregation techniques: bagging [4] and boosting [14]. The theoretical framework on aggregation is an active field of research and is developing fast [5, 8, 17]. Bagging uses a majority vote to aggregate the component classifiers with each component classifier being given an equal weight, i.e. $\alpha_l = \frac{1}{L}$ for $l \in \{1, ..., L\}$. A drawback of bagging is that the weights are fixed independently of the design data set, i.e. a bad component classifier is given the same weight as a good component classifier. In AdaBoost.M1, the weight of the l-th component classifier is chosen as $\alpha_l = \log((1 - e_l)/e_l)$ where e_l is the fraction of design data set patterns that are misclassified by component classifier g_l. Notice that in AdaBoost.M1, the weights α_l can be negative and it is not required that at least one $\alpha_l > 0$, $l \in \{1, ..., L\}$. Boosting takes the design data set indirectly into account by letting the weight α_l depend on e_l. An alternative to bagging and boosting is to use mathematical programming approaches to determine the values for the weights α_l according to some criterion function defined directly on the available design data set [2]. Instead of finding an aggregate classifier by computing weights and performing a majority vote, we show that enumeration over all possible distinct aggregate classifiers is a viable alternative, especially when the number of component classifiers is not too large.

In our setting, the aggregation problem can be formulated as follows:

> [**AP**] For each design data set pattern n, L component classifications $g_l(\mathbf{x}_n) \in \{1, ..., C\}$ and the true class $c_n \in \{1, ..., C\}$ are given, $n \in \{1, ..., N\}$. Find weights α_l such that $\alpha_l \geq 0$ and at least one $\alpha_l > 0$, $l \in \{1, ..., L\}$.

There is no objective function specified in **AP**. We will focus on counting the number of solutions to **AP** and, hence, the results are valid irrespective of the objective function.

Notice however that an obvious objective function would be to minimize the number of misclassifications on the design data set. This problem is referred to as **AP-MIN**. It can be shown that **AP-MIN** is NP-hard [1].

Our contribution is threefold. First, we propose a new way to find a best aggregate classifier. Second, the question of counting the number of distinct aggregate classifiers is partially answered. Third, the link between aggregating classifiers for $C = 2$ and n-bit boolean functions is described which allows a reinterpretation of the aggregate classifier count in terms of n-bit boolean functions.

In the second section, the solution space of **AP** is studied and the framework to count the number of solutions to **AP** is built. The third section presents an algorithm to count the number of solutions to **AP**. In section 4, the link between n-bit boolean functions and aggregating classifiers for $C = 2$ is discussed. The last section summarizes the conclusions.

2 Analyzing the Solution Space of AP

First, a solution to **AP** is defined.

Definition 1. An L-tuple $(\alpha_1,...,\alpha_L)$ is a *solution* to **AP** if and only if

(i) $\alpha_l \geq 0$ and $\in$ IR with $l \in \{1, ..., L\}$ and

(ii) for every possible partition of the set $\{g_1, ..., g_L\}$ into at most $\min\{L, C\}$ nonempty subsets, there exists a subset $\mathcal{G}^*$ in the partition such that $\sum_{g_l \in \mathcal{G}^*} \alpha_l > \sum_{g_l \in \mathcal{G}} \alpha_l$ with $\mathcal{G}$ any other subset of the partition.

Notice that conditions (i) and (ii) imply that (0,...,0) is excluded as a solution, which comes down to saying that, in order to have a meaningful aggregation, at least one component classifier must receive a strictly positive weight. Condition (i) forbids strictly negative weights as only positive voting is allowed, i.e. we allow only for votes in favor of a class rather than against one. By condition (ii), the aggregate classifier will also be a function, i.e. ties are excluded. To see this, define the C sets $\mathcal{G}_{nc} = \{g_l \,|\, g_l(\mathbf{x}_n) = c\}$, $c \in \{1, ..., C\}$ for a given design data set pattern n and consider only the nonempty sets $\mathcal{G}_{nc}$. These nonempty sets form a partition of the set $\{g_1, ..., g_L\}$ into at most $\min\{L, C\}$ nonempty subsets. Conversely, every partition of the set $\{g_1, ..., g_L\}$ into at most $\min\{L, C\}$ nonempty subsets represents a way in which the classifications of the component classifiers can differ, i.e. all component classifiers that are in the same subset of such partition give the same classification for $\mathbf{x}_n$ which differs from the classification of component classifiers in any other subset of the partition. Condition (ii) states that the weights should be determined in such a way that there is always a unique aggregation decision, i.e. for all the possible ways in which the classifications of the component classifiers can differ. Notice that it is not relevant which value from $\{1, ..., C\}$ is associated to which nonempty subset in the partition. What matters in the analysis of the solution space is the mechanism of the aggregation, not the outcome.

> **Example 2.** Let $L = 3$ and $C = 4$. The 3-tuple $(\alpha_1,\alpha_2,\alpha_3) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ is not a solution, as condition (ii) is not satisfied. Indeed, for the partition $\{\{g_1\},\{g_2\},\{g_3\}\}$, there would be a tie. Notice that $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ is the bagging solution. Some examples of solutions are (1,0,0), $(\frac{7}{10}, \frac{2}{10}, \frac{1}{10})$ and (4,3,3). ■

Let $\mathcal{T}$ be the set of all the possible partitions of the set $\{g_1, ..., g_L\}$ into at most $\min\{L, C\}$ nonempty subsets. Order the $|\mathcal{T}|$ elements of $\mathcal{T}$ into any order and label them from 1 to $|\mathcal{T}|$. Denote by $\mathcal{P}_t$ the t-th element of $\mathcal{T}$, $t \in \{1, ..., |\mathcal{T}|\}$ with

$$|\mathcal{T}| = \sum_{k=1}^{\min\{L,C\}} S(L, k)$$

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 [6, 19].

Let $(\alpha_1,...,\alpha_L)$ be a solution to **AP** and denote by $\mathcal{G}_t^*$ the (unique) subset in $\mathcal{P}_t$ for which $\sum_{g_l \in \mathcal{G}_t^*} \alpha_l > \sum_{g_l \in \mathcal{G}_t} \alpha_l$ with $\mathcal{G}_t$ any other subset of $\mathcal{P}_t$, $t \in \{1, ..., |\mathcal{T}|\}$. Condition (ii) in Definition 1 implies that with every L-tuple that is a solution, exactly one $|\mathcal{T}|$-dimensional vector $[\mathcal{G}_1^* \ ... \ \mathcal{G}_{|\mathcal{T}|}^*]$ can be associated. This $|\mathcal{T}|$-dimensional vector will be called the *partition decision vector* of the solution as it represents the aggregation decision, in terms of nonempty subsets, for all $\mathcal{P}_t$ in $\mathcal{T}$.

In order to structure the solution space of **AP** further, a second definition is introduced.

> **Definition 2.** Let the L-tuples $(\alpha_1,...,\alpha_L)$ and $(\beta_1,...,\beta_L)$ both be solutions. $(\alpha_1,...,\alpha_L)$ and $(\beta_1,...,\beta_L)$ are *distinct solutions* if and only if their partition decision vectors are not the same.

To see the intuition of the definition, notice that the weights are just a means to merge the (possibly) different classifications of the component classifiers into a (unique) aggregated prediction. Hence, if two distributions of weights are such that they always, i.e. for all $\mathcal{P}_t$ in $\mathcal{T}$, yield the same aggregate decision, they can be considered identical.

> **Example 2 (continued).** The 3-tuples (1,0,0), $(\frac{7}{10}, \frac{2}{10}, \frac{1}{10})$ and (4,3,3) are solutions. (1,0,0) and (4,3,3) are distinct solutions as for partition $\{\{g_1\}, \{g_2, g_3\}\}$, the aggregate decision according to (1,0,0) is $\{g_1\}$, but according to (4,3,3) it is $\{g_2, g_3\}$. By exhaustively going through all cases, $\{\{g_1, g_2, g_3\}\}$, $\{\{g_1\}, \{g_2, g_3\}\}$, $\{\{g_2\}, \{g_2, g_3\}\}$, $\{\{g_3\}, \{g_2, g_3\}\}$ and $\{\{g_1\}, \{g_2\}, \{g_3\}\}$, it can be verified that no partitions of at most three nonempty subsets exist for which (1,0,0) and $(\frac{7}{10}, \frac{2}{10}, \frac{1}{10})$ yield a different aggregate decision and hence, they are not distinct solutions. ■

The above definitions structure the solution space of **AP**. Let $\mathcal{S}_{\text{sol}} = \{ (\alpha_1, ..., \alpha_L) \mid (\alpha_1, ..., \alpha_L) \text{ is a solution}\}$. For all values of L and C, the set $\mathcal{S}_{\text{sol}}$ contains infinitely many elements, $L \in \{1, 2, ...\}$, $C \in \{2, 3, ...\}$. Definition 2 partitions the set $\mathcal{S}_{\text{sol}}$ into Q subsets $\mathcal{S}^q_{\text{sol}}$ with $q \in \{1, ..., Q\}$ in such a way that (i) any two solutions that belong to the same subset are not distinct, and (ii) any two solutions that belong to different subsets are distinct. A visualization of the structure of the solution space of **AP** is given in Figure 1.

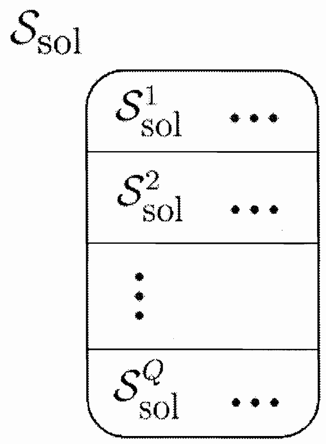
$\mathcal{S}_{\text{sol}}$
$\mathcal{S}^1_{\text{sol}}$ ...
$\mathcal{S}^2_{\text{sol}}$ ...
⋮
$\mathcal{S}^Q_{\text{sol}}$ ...

Figure 1: Visualization of the Structure of the Solution Space of **AP**

Having introduced the necessary elements for the analysis of the solution space of **AP**, Q can be formally defined.

> **Definition 3.** Let C be a finite number in $\{2, 3, ...\}$ and L be a finite number in $\{1, 2, ...\}$. $Q(L, C)$ is the number of distinct solutions of **AP** with L component classifiers and C classes.

It is easy to see that, for all possible values of L and C, $Q(L, C)$ is finite. Definition 2 implies that the number of distinct solutions to **AP** cannot be larger than the number of different partition decision vectors, which is obviously finite for finite values of L and C.

Hence, a representation of the solution space of **AP** is obtained that is discrete rather than continuous. This is an important finding as it reveals the discrete nature of **AP**. In terms of solution methods for **AP**, it means that any instance of **AP** with given L and C can be solved by enumeration over $Q(L, C)$ distinct solutions, in the worst case. Notice that this is true irrespective of the objective function.

There are still two important unanswered questions.

> **Question 1.** How large is $Q(L, C)$?

> **Question 2.** Given $Q(L, C)$, how are $Q(L, C)$ distinct solutions obtained?

In the next section, these questions will be partially answered. We are not able to come up with a direct or recursive expression for $Q(L, C)$ as a function of L and C. As a best alternative, an algorithm to count $Q(L, C)$ will be presented. Interestingly, the counting algorithm will also provide an answer to question 2.

3 An Algorithm to Compute $Q(L, C)$

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

$$\prod_{t=1}^{|\mathcal{T}|} |\mathcal{P}_t|$$

which can be rewritten as

$$\prod_{k=1}^{\min\{L,C\}} k^{S(L,k)}$$

since $|\mathcal{P}_t| \in \{1, ..., \min\{L,C\}\}$ for $t \in \{1, ..., |\mathcal{T}|\}$. $S(L,k)$ is again the Stirling number of the second kind, or, the number of ways to partition a set of L elements into exactly k nonempty subsets.

The number of candidate partition decision vectors for a given value of L and C gives a (loose) 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.

In Table 1, the number of candidate partition decision vectors for small **AP** are given. The numbers get very large very fast. E.g. for L=5 and C=3, the result is $2^{15}3^{25}$ which is already a 17-digit number.

Table 1: Number of Candidate Partition Decision Vectors for Small **AP**

	C=2	C=3	C=4	C=5
$L = 1$	2^0	2^0	2^0	2^0
$L = 2$	2^1	2^1	2^1	2^1
$L = 3$	2^3	$2^3 3^1$	$2^3 3^1$	$2^3 3^1$
$L = 4$	2^7	$2^7 3^6$	$2^7 3^6 4^1$	$2^7 3^6 4^1$
$L = 5$	2^{15}	$2^{15} 3^{25}$	$2^{15} 3^{25} 4^{10}$	$2^{15} 3^{25} 4^{10} 5^1$

By Definition 2, it suffices to generate all candidate partition decision vectors and check if there exists a solution that could give the candidate partition decision vector. If so, the candidate partition decision vector is a partition decision vector. The check comes down to solving the following feasibility problem:

[**FP**] Is the system of linear strict inequalities $\{\sum_{g_l \in \mathcal{G}_t^*} \alpha_l - \sum_{g_l \in \mathcal{G}_t} \alpha_l > 0, \mathcal{G}_t \neq \mathcal{G}_t^*, t \in \{1, ..., |\mathcal{T}|\}\}$ feasible for real valued nonnegative $\alpha_l, l \in \{1, ..., L\}$?

As solutions can be multiplied by a real number $r > 0$ without changing the aggregate classifications, **FP** is equivalent to **FP-**ϵ which can be solved in polynomial time by linear programming techniques. These techniques also yield a feasible solution if there exists one and hence, in this way, answer Question 1 and 2 at the same time and same computational cost.

[**FP-**ϵ] Let $\epsilon \in$ IR and > 0. Is the system of linear inequalities $\{\sum_{g_l \in \mathcal{G}_t^*} \alpha_l - \sum_{g_l \in \mathcal{G}_t} \alpha_l \geq \epsilon, \mathcal{G}_t \neq \mathcal{G}_t^*, t \in \{1, ..., |\mathcal{T}|\}\}$ feasible for real valued nonnegative $\alpha_l, l \in \{1, ..., L\}$?

However, **FP-**ϵ needs to be solved for *all* candidate partition decision vectors and, as is illustrated in Table 1, the number of candidate partition decision vectors gets very large very fast. However, there is a way to reduce the number of candidate partition decision vectors for which **FP-**ϵ needs to be solved.

If the following consistency rule does not hold, we know beforehand that **FP-**ϵ cannot be feasible.

Consistency Rule. Consider $\mathcal{P}_t$ and let $\mathcal{G}_t^*$ be the t-th entry in the candidate partition decision vector. Then, for all $\mathcal{P}_s$, $s \in \{1, ..., |\mathcal{T}|\}$, $s \neq t$ for which

(i) there exists a subset $\mathcal{G}_s^*$ such that $\mathcal{G}_t^* \subset \mathcal{G}_s^*$ and,

(ii) every other subset $\mathcal{G}_s$ of $\mathcal{P}_s$ is a subset of some other subset $\mathcal{G}_t$ of $\mathcal{P}_t$,

it must hold that $\mathcal{G}_s^*$ is the s-th entry in the candidate partition decision vector, for otherwise the candidate partition decision vector cannot be a partition decision vector.

The idea behind the rule is consistency. Loosely speaking, if, for a partition, there is a subset of α_l's that dominates the other subsets, it has to dominate those subsets also in the other partitions.

Example 3. Let $L = 3$ and $C = 4$. There are five possible partitions of at most $\min\{3,4\}$ nonempty subsets. Suppose we are constructing two candidate partition decision vectors by filling in the entries. Below, it can be seen that for the first candidate partition decision vector, only the second entry has been determined and for the second, only the fifth position has been filled up.

	Construction of Candidate Partition Decision Vector 1	Construction of Candidate Partition Decision Vector 2
$\{\{g_1, g_2, g_3\}\}$	?	?
$\{\{g_1, g_2\}, \{g_3\}\}$	$\{g_3\}$	?
$\{\{g_1, g_3\}, \{g_2\}\}$	?	?
$\{\{g_2, g_3\}, \{g_1\}\}$	?	?
$\{\{g_1\}, \{g_2\}, \{g_3\}\}$	?	$\{g_1\}$

In the first candidate partition decision vector, the second entry $\{g_3\}$ implies that $\alpha_1 + \alpha_2 < \alpha_3$. By the consistency rule, this choice determines *all* further entries. E.g., if the candidate partition decision vector would have g_2 as a third entry, it cannot be a partition decision vector as no 3-tuple (α_1,α_2,α_3) can be found for which both $\alpha_1 + \alpha_2 < \alpha_3$ and $\alpha_1 + \alpha_3 < \alpha_2$. In the second candidate partition decision vector, the entry $\{g_1\}$ determines the entry for $\{\{g_1, g_2, g_3\}\}$, $\{\{g_1, g_2\}, \{g_3\}\}$ and $\{\{g_1, g_3\}, \{g_2\}\}$. For $\{\{g_2, g_3\}, \{g_1\}\}$, condition (ii) of the consistency rule is not fulfilled. ■

Rather than enumerating all the possible candidate partition 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. Table 2 gives the number of candidate partition decision vectors that comply with the consistency rule. It should be compared to Table 1. E.g. for $L = 4$ and $C = 3$, only 116 feasibility checks are required while the number of candidate partition decision vectors is $2^7 3^6 = 93312$.

Table 2: Number of Candidate Partition Decision Vectors that Comply with the Consistency Rule for Small **AP**

	C=2	C=3	C=4	C=5
L=1	1	1	1	1
L=2	2	2	2	2
L=3	4	6	6	6
L=4	12	116	140	140
L=5	81	756865	2503042	2867234

Below, an informal description is given of an algorithm that counts the number of distinct solutions for **AP**, or, the number of subsets $\mathcal{S}^q_{\text{sol}}$, or, $Q(L, C)$ for a given value of L and C, $L \in \{1, 2, ...\}$, $C \in \{2, 3, ...\}$. This algorithm will be referred to as the counting algorithm.

```
Input: A number $L \in \{1,2,...\}$, a number $C \in \{2,3,...\}$.

    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 partition
       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 partition decision vector,
       $\mathcal{G}^*_t = \mathcal{G}_t$.

    4. If possible, fill in unfilled entries of the to-be-built
       candidate partition decision vector by application of the
       consistency rule.

    5. If all the entries of the to-be-built candidate partition
       decision vector are filled in, a candidate partition
       decision vector has been built and proceed to step 6;
       else update $t$ to the next unfilled entry in the to-be-built
       candidate partition decision vector and go to step 3.

    6. Use FP-$\epsilon$ to check if there exists a solution that would
       give the candidate partition decision vector. If so, the
```

```
   candidate partition decision vector is a partition decision
   vector, $q \leftarrow q+1$ and the solution is saved.

7. Empty the entries of the to-be-built candidate partition
   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.
```

In Table 3, the values for $Q(L,C)$ for small **AP** 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.

Table 3: $Q(L,C)$ for Small **AP**

	C=2	C=3	C=4	C=5	C=6
L=1	1	1	1	1	1
L=2	2	2	2	2	2
L=3	4	6	6	6	6
L=4	12	76	84	84	84
L=5	81	7625	13805	14025	14025
L=6	1684	?	?	?	?
L=7	122921	?	?	?	?
L=8	?	?	?	?	?

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 counting algorithm provides us with a set of $Q(L,C)$ solutions, each of which is associated to a distinct aggregate classifier. Consequently, in these cases, the aggregation problem can be solved by enumeration over the set of $Q(L,C)$ solutions, whatever the objective function. Notice that we only need to run the counting algorithm once to obtain a set of $Q(L,C)$ solutions. This set then suffices to solve any instance of the aggregation problem with L component classifiers and C classes. In contrast to the mathematical programming approach in [2], the influence on the computational performance of the number of patterns N in the design data set is negligible.

> **Example 4.** Take any instance of **AP** with $L = 3$ and $C = 4$. If follows from the counting algorithm that $Q(3,4) = 6$. Hence, there are only six subsets $\mathcal{S}^q_{\text{sol}}$ and from each of these six subsets, which contain infinitely many elements, only one element needs to be checked e.g. (1,0,0), (0,1,0), (0,0,1), (4,3,2) (2,4,3), (2,3,4). Whatever the criterion function, the instance will be solved to optimality by calculating the criterion function value for these six 3-tuples and choosing the one with the highest value of the criterion function. ■

Instead of finding an aggregate classifier by computing weights and performing a majority vote, we show that enumeration over all possible distinct aggregate classifiers is a viable alternative, especially when the number of component classifiers is not too large.

Notice that $Q(L,C) = Q(L,L)$ whenever $C > L$, $L \in \{1,...,5\}$. This is not surprising as it is not difficult to see that when $C > L$, $Q(L,C)$ depends on L only. [16] provides an on-line encyclopedia of integer sequences. Interestingly, none of the sequences in the table (e.g. 2,6,84,14025 or 1,2,4,12,81,1684,122921) can be found in the on-line encyclopedia. Also the question whether or not a recursive relation for $Q(L,C)$ exists remains open.

From Table 3, it is clear that the numbers $Q(L,C)$ grow extremely fast with increasing L and, due to limited computational resources, one will never be able to find $Q(L,C)$ when both L gets large, say $L = 100$, by means of the counting algorithm. However, for small aggregation problems, i.e., those for which it is possible to obtain $Q(L,C)$ by means of the counting algorithm within a reasonable amount of computing time, exact solutions can be obtained whatever the objective function. 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.

In the next section, the link between **AP** and n-bit boolean functions is described. The main motivation for presenting the link is that it yields a new way of looking at **AP** and the problem of computing $Q(L, C)$.

4 Linking AP with $C = 2$ to n-Bit Boolean Functions

Let $n \in \{1, 2, ...\}$. An n-bit boolean function is a function of the form $b : \{0, 1\}^n \mapsto \{0, 1\} : \mathbf{x} \mapsto b(\mathbf{x})$. The domain $\{0, 1\}^n$ contains 2^n elements. Hence, the number of n-bit boolean functions is 2^{2^n} [11]. We refer to [7] for an introduction to boolean functions.

Example 5. Say $n = 2$. The $2^{2^2} = 16$ 2-bit boolean functions are shown in Table 4. ■

Table 4: 2-Bit Boolean Functions

x_1	x_2	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}	b_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

To see the link with **AP**, assume $C = 2$ and the two classes are coded by 0 and 1. Then, for $C = 2$ and $L \in \{1, 2, ...\}$, each aggregated classifier that is given by a solution of **AP** can be seen as a function from $\{0, 1\}^L \mapsto \{0, 1\}$ and consequently, must correspond to one of the 2^{2^L} possible n-bit boolean functions, where $n = L$. The words "boolean" and "bit" simply refer to the fact that $C = 2$. More general, **AP** with $C \in \{2, 3, ...\}$ and $L \in \{1, 2, ...\}$ can be linked with the set of functions, $b : \{0, 1, ..., C - 1\}^L \mapsto \{0, 1, ..., C - 1\} : (x_1, ..., x_L) \mapsto b(x_1, ..., x_L)$. In the remainder of this section, the focus is on the case $C = 2$.

Example 5 (continued). Let $C = 2$ and $L = 2$. It is easy to verify that $(\alpha_1,\alpha_2) = (0.7,0.3)$ is a solution to **AP**. The aggregated classifier that is associated to (0.7,0.3) gives the following classifications,

$g_1(\mathbf{x})$	$g_2(\mathbf{x})$		$\arg\max_{c\in\{0,1\}}\{\sum_{l=1}^{2}\alpha_l I(g_l(\mathbf{x})=c)\}$
0	0	$\rightarrow$	0
0	1	$\rightarrow$	0
1	0	$\rightarrow$	1
1	1	$\rightarrow$	1

Hence, it corresponds to the 2-bit boolean function b_4. ■

For $C = 2$ and $L \in \{1, 2, ...\}$, each solution to **AP** corresponds to exactly one n-bit boolean function where $n = L$. The reverse is not true. There are n-bit boolean functions that do not correspond to a solution to **AP** with $C = 2$ and $L = n$. Also, every n-bit boolean function that corresponds to at least one solution to **AP** with $C = 2$ and $L = n$, corresponds to an infinite number of such solutions. Further, distinct solutions to **AP** correspond to different n-bit boolean functions. Solutions to **AP** that are not distinct correspond to the same n-bit boolean function. The proofs of these claims are simple and are left out (see [1]).

This establishes a relation between the set of solutions $\mathcal{S}_{\text{sol}}$ for **AP** with $C = 2$ and $L \in \{1, 2, ...\}$ and the set of n-bit boolean functions with $L = n$. The relation is visualized in Figure 2.

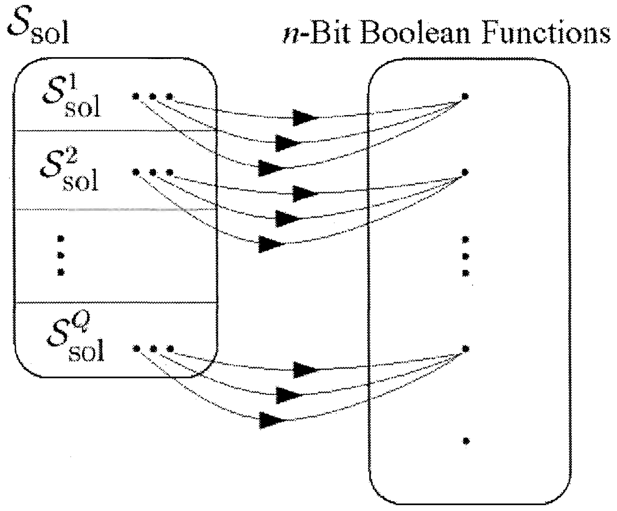

Figure 2: Visualization of the Relation between the Set of Solutions of **AP** with $C = 2$ and $L = n$ and the Set of n-Bit Boolean Functions

From the relation, it follows that counting the number of n-bit boolean functions that correspond to at least one solution of **AP** with $C = 2$ and $L = n$ is an alternative way to count $Q(L, 2)$. Hence, the problem of computing $Q(L, 2)$ can also be formulated in terms of n-bit boolean functions.

Let $\mathbf{X} \in \{0,1\}^{2^n \times n}$ be the matrix that consists of all the possible inputs of an n-bit boolean function. Denote by $\mathbf{x}_i$ the i-th row of $\mathbf{X}$, $i \in \{1, ..., 2^n\}$. Let $\mathbf{A} \in \{-1,1\}^{2^n \times n}$ and $\mathbf{y} \in \mathrm{IR}^{n \times 1}$.

Definition 4. An n-bit boolean function $b : \{0,1\}^n \mapsto \{0,1\} : \mathbf{x} \mapsto b(\mathbf{x})$ is an *aggregate n-bit boolean function* if the system $\{\mathbf{Ay} > \mathbf{0}, \mathbf{y} \geq \mathbf{0}\}$ has a feasible solution where $a_{ij} = 1$ if $x_{ij} = b(\mathbf{x}_i)$ and $a_{ij} = -1$ if $x_{ij} \neq b(\mathbf{x}_i)$, $i \in \{1, ..., 2^n\}$ and $j \in \{1, ..., n\}$.

We claim that every aggregate n-bit boolean function corresponds to at least one solution of **AP** with $C = 2$ and $L = n$ and, consequently, computing $Q(L, 2)$ can be done by counting the number of aggregate n-bit boolean functions with $n = L$. Before proving this claim, two properties of aggregate n-nit boolean functions will be presented.

Let x be an element of $\{0,1\}$. If x=1, then $\bar{x}$=0 and if x=0, then $\bar{x}$=1. An n-bit boolean function is *self-dual* if for all $(x_1,...,x_n) \in \{0,1\}^n$, $b(x_1,...,x_n) = \bar{b}(\bar{x}_1,...,\bar{x}_n)$. It is easy to see that the number of self-dual n-bit boolean functions is $2^{2^{n-1}}$.

Example 5 (continued). There are $2^{2^{2-1}} = 2^2 = 4$ self-dual 2-bit boolean functions. They are shown in Table 5. ■

Table 5: Self-Dual 2-Bit Boolean Functions

x_1	x_2	b_4	b_6	b_{11}	b_{13}
0	0	0	0	1	1
0	1	0	1	0	1
1	0	1	0	1	0
1	1	1	1	0	0

This leads to the following proposition.

Proposition 1. An aggregate n-bit boolean function is self-dual.

Proof : For any n-bit boolean function, let $\mathbf{X} \in \{0,1\}^{2^n \times n}$ be the matrix that consists of all the possible inputs of an n-bit boolean function. Let $\mathbf{x}_i$ be the i-th row of $\mathbf{X}$, $i \in \{1,...,2^n\}$. For every row $\mathbf{x}_i$, $i \in \{1,...,2^n\}$, there exists a row $\mathbf{x}_j$, $j \in \{1,...,2^n\}$, $i \neq j$, such that for all $k \in \{1,...,n\}$, $x_{jk} = 1 - x_{ik} = \bar{x}_{ik}$. If $b(\mathbf{x}_i) = b(\mathbf{x}_j)$, $\{\mathbf{Ay} > \mathbf{0}, \mathbf{y} \geq \mathbf{0}\}$ cannot be feasible as inequality i and j will be contradictory. Hence, if b is an aggregate n-bit boolean function, it must hold that $b(\mathbf{x}_i) \neq b(\mathbf{x}_j)$, or $b(\mathbf{x}_i) = \bar{b}(\bar{\mathbf{x}}_i)$. As this is true for all $i \in \{1,...,2^n\}$, an aggregate n-bit boolean function is self-dual. (Q.E.D.)

Let $\mathcal{B}_0 = \{b \,|\, b(0,...,0) = 0\}$. The number of self-dual n-bit boolean functions that are an element of $\mathcal{B}_0$ is $2^{2^{n-1}-1}$ which equals the number of candidate partition decision vectors for $C = 2$ and $L = n$, i.e. $\prod_{k=1}^{\min\{L,2\}} k^{S(L,k)} = 2^{2^{n-1}-1}$ (see also Table 1).

For any $(x_1,...,x_n),(y_1,...,y_n) \in \{0,1\}^n$: $(x_1,...,x_n) \leq (y_1,...,y_n)$ when $x_i \leq y_i$ for $i \in \{1,...,n\}$. An n-bit boolean function is *monotone* if $b(x_1,...,x_n) \leq$

$b(y_1, ..., y_n)$ whenever $(x_1, ..., x_n) \leq (y_1, ..., y_n)$. Determining the number of monotone n-bit boolean functions is known as Dedekind's problem. Many mathematicians contributed to this problem but despite these efforts, only for small values of n the number of monotone n-bit boolean functions is known [12].

Example 5 (continued). The 6 monotone 2-bit boolean functions are shown in Table 4. ■

Table 6: Monotone 2-Bit Boolean Functions

x_1	x_2	b_1	b_2	b_4	b_6	b_8	b_{16}
0	0	0	0	0	0	0	1
0	1	0	0	0	1	1	1
1	0	0	0	1	0	1	1
1	1	0	1	1	1	1	1

This leads to the following proposition.

Proposition 2. An aggregate n-bit boolean function is monotone.

Proof : Assume there exists an aggregate n-bit boolean function b that is not monotone. Then, there exists an $(x_1, ..., x_n)$ and $(y_1, ..., y_n)$ such that $(x_1, ..., x_n) \leq (y_1, ..., y_n)$ and $b(x_1, ..., x_n) > b(y_1, ..., y_n)$. Hence, $b(x_1, ..., x_n) = 1$ and $b(y_1, ..., y_n) = 0$. As $(x_1, ..., x_n) \leq (y_1, ..., y_n)$, adding up the inequalities in the system $\{\mathbf{Ay} > \mathbf{0}, \mathbf{y} \geq \mathbf{0}\}$ that correspond to $(x_1, ..., x_n)$ and $(y_1, ..., y_n)$ leads to a contradiction. If $(x_1, ..., x_n) < (y_1, ..., y_n)$, the contradiction is established by the fact that $\mathbf{y} \geq \mathbf{0}$. (Q.E.D.)

By Proposition 1 and 2, the number of self-dual monotone n-bit boolean functions is an upper bound for the number of aggregate n-bit boolean functions. The number of self-dual monotone n-bit boolean functions is not easy to determine in general and known as sequence A001206 in the on-line encyclopedia of integer sequences [16].

n	number of self-dual monotone n-bit boolean functions
1	2
2	4
3	12
4	81
5	2646
6	1422564
7	229809982112

Without proof, we notice that the number of self-dual monotone n-bit boolean functions is exactly the the number of candidate partition decision vectors that comply with the consistency rule for $C = 2$ and $L = n$. For $C = 2$ the counting algorithm simply generates all self-dual monotone boolean n-bit functions and, only for those, checks whether or not they are an aggregate boolean function by solving **FP-**ϵ.

Example 5 (continued). The two self-dual monotone 2-bit boolean functions are shown in Table 7.

Table 7: Monotone Self-Dual 2-Bit Boolean Functions

x_1	x_2	b_4	b_6
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

Notice that, in this case, both monotone self-dual 2-bit boolean functions can be associated to a solution of to **AP** for $C = 2$ and $L = 2$. E.g. b_4 can be associated to (0.7, 0.3) and b_6 to (0.3, 0.7), which are two distinct solutions to **AP** for $C = 2$ and $L = 2$. ■

The main result will now be proven.

Proposition 3. An aggregate n-bit boolean function corresponds to at least one solution of **AP** with $C = 2$ and $L = n$.

Proof : Let $\mathbf{y} = (y_1, ..., y_n)$ be a feasible solution of the system $\{\mathbf{Ay} > \mathbf{0}, \mathbf{y} \geq \mathbf{0}\}$. Hence, $\mathbf{y}$ is an L-tuple that satisfies condition (i) of Definition 1 and $\sum_{i=1}^{n} y_i > 0$, for otherwise it cannot be true that $\mathbf{Ay} > \mathbf{0}$. Two cases are possible.

(i) $L = 1$. If y_1 satisfies condition (i) of Definition 1 and $y_1 > 0$, y_1 must also satisfy condition (ii) as, in this case, $|\mathcal{T}| = 1$.

(ii) $L \geq 2$. Assume condition (ii) of Definition 1 is not satisfied. As $C = 2$, this means that there exists a $\mathcal{P}_t$ that consists of two nonempty subsets, say $\mathcal{G}^0$ and $\mathcal{G}^1$, such that $\sum_{g_l \in \mathcal{G}^0} y_l = \sum_{g_l \in \mathcal{G}^1} y_l$. In that case there are two strict inequalities in $\{\mathbf{Ay} > \mathbf{0}, \mathbf{y} \geq \mathbf{0}\}$ that are not satisfied. This contradicts that $\mathbf{y}$ is a feasible solution to the system $\{\mathbf{Ay} > \mathbf{0}, \mathbf{y} \geq \mathbf{0}\}$. Hence, $\mathbf{y}$ must satisfy condition (ii) in Definition 1 and is a solution to **AP** with $C = 2$ and $L = n$.(Q.E.D.)

To summarize, determining $Q(L, 2)$ for $L \in \{1, 2, ...\}$ boils down to counting the number of aggregate n-bit boolean functions for $n = L$. To count the number of aggregate boolean functions, it suffices to see how many of the self-dual monotone n-bit boolean functions are aggregate n-bit boolean functions. The link between n-bit boolean functions and **AP** for $C = 2$ and $L = \{1, 2, ...\}$ indicates that this is exactly what the counting algorithm of Section 3 does. In a sense, the link also justifies the counting algorithm as for similar counting problems in the field of n-bit boolean functions, one has to resort to simple counting algorithms too.

5 Conclusions

By studying the solution space of the aggregation problem, it was shown that number of distinct solutions of the aggregation problem is finite. An algorithm is presented that is able to count and save the distinct solutions, which makes it possible to find exact solutions to the aggregation problem by simple enumeration whatever the objective function. Interestingly, the sequence of the number of distinct solutions appears to be new. Evidently, for lack of computational power, this approach is only successful for instances of the aggregation problem where L and C are small. There is a link between the aggregation problem for $C = 2$ and n-bit boolean functions which provides a different way of looking at the aggregation problem and opens up new opportunities to study the aggregation problem.

Acknowledgement

We thank Yves Crama and Peter Goos for their helpful comments.

References

[1] Adem J. (2004) Mathematical Programming Approaches for the Supervised Classification Problem, PhD Thesis, Katholieke Universiteit Leuven.

[2] Adem J. and Gochet W. (to appear) *Aggregating Classifiers with Mathematical Programming* Computational Statistics and Data Analysis.

[3] Amaldi E. and Kann V. (1995) *The Complexity and Approximability of Finding Maximum Feasible Subsystems of Linear Relations* Theoretical Computer Science 147 (1-2) pp 181-210.

[4] Breiman L. (1996) *Bagging Predictors* Machine Learning 24 pp 123-140.

[5] Bühlmann P. and Yu B. (1996) *Analyzing Bagging* The Annals of Statistics 30 (4) pp 927-961.

[6] Conway J. and Guy, R. (1996) The Book of Numbers, Springer-Verlag New York.

[7] Crama Y. and Hammer P. (in preparation) Boolean Functions Theory, Algorithms, and Applications `http://www.eaa.egss.ulg.ac.be/rogp/crama/`.

[8] Friedman J., Hastie T. and Tibshirani R. (2000) *Additive Logistic Regression: a Statistical View of Boosting* The Annals of Statistics 28 (2) pp 337-374.

[9] Hastie T., Tibshirani R. and Friedman J. (2001) The Elements of Statistical Learning, Springer.

[10] Höffgen K. and Simon H. (1995) *Robust Trainability of Single Neurons* Journal of Computer and System Sciences 50 (1) pp 114-125.

[11] Kuntzmann J. (1967) *Fundamental Boolean Algebra* Blackie London

[12] Mathpages (2003) Dedekind's Problem, published electronically at `http://www.mathpages.com/home/kmath030.htm`.

[13] Muroga S., Tsuboi T. and Baugh C. (1970) *Enumeration of Threshold Functions of Eight Variables* IEEE Transactions on Computers 19 (9) pp 818-825.

[14] Schapire R. (1990) *The Strength of Weak Learnability* Machine Learning 5 (2) pp 197-227.

[15] Schapiro H. (1970) *On the Counting Problem for Monotone Boolean Functions* Communications on Pure and Applied Mathematics 23 pp 299-312.

[16] Sloane N. (2003) *The On-Line Encyclopedia of Integer Sequences* published electronically at `http://www.research.att.com/~njas/sequences/`.

[17] Tsybakov A. (2004) *Optimal Aggregation of Classifiers in Statistical Learning* The Annals of Statistics 32 (1) pp 135-166.

[18] Webb A. (1999) Statistical Pattern Recognition, Arnold London.

[19] Weisstein E. (2004) Stirling Number of the Second Kind, from MathWorld - A Wolfram Web Resource published electronically at `http://mathworld.wolfram.com/StirlingNumberoftheSecondKind.html`.