Robust Multiclass Ensemble Classifiers via Symmetric Functions

Patrice Lefaucheur* Richard Nock*

Abstract

We introduce a generalization to the multiclass framework of a previous approach to Boosting by constructing symmetric functions. This approach contrasts with the usual ADBOOST-type Boosting algorithms using linear separators. Indeed, multiclass induction does not necessitate combination tricks such as those for linear separators, and it achieves some novel agnostic learning properties, as well as significant malicious noise tolerance. Experiments on a large testbed against ADBOOST and C4.5 display the efficiency of the approach pruned.

Keywords: Ensemble classifiers, Symmetric functions.

1 Introduction

Advances in Machine Learning (ML), Data Mining (DM) and Classification have shown experimentally and theoretically the power of ensemble methods, that is, algorithms combining the predictions of multiple classifiers to make a single classifier [1]. Most of these approaches are voting procedures [3]. Boosting refers to a broader class of algorithms, not restricted to voting. However, boosting has been used so many times for building linear separators that it has been defined as such: "Boosting is a method of sequential linear combination of functions in a base hypothesis space" [4]. This definition, which is a voting-like approach of boosting, significantly narrows its scope. Indeed, boosting only refers to the ability an algorithm has for using the formulas \(h_1, h_2, \ldots, h_T\) output by a probably moderately accurate algorithm, to build with high probability a combination \(H_T\) of them having arbitrary high accuracy (with \(T\) large enough) [6]. It does not necessarily state that \(H_T\) is a linear combination (nor any other particular formula), even if this habit is probably historically rooted in the first conjectures about the existence of boosting algorithms, and later in their numerous theoretical and experimental successes [9].

A recent work has departed from the boosting main-stream [7], in that it shows that one can keep boosting properties while replacing linear separators by symmetric functions (SF), i.e. functions whose output is invariant under permutation of the input bits. This has the main benefit to bring strong noise handling while boosting, a property hardly achievable with linear separators [1]. However, the result of [7] has a restricted scope: it only studies classification problems with two classes. Most hard classification problems have more than two classes, and fitting multiclass problems to two classes is everything but trivial [9].

In this paper, we propose a strict generalization of the framework of [7] and their algorithm SFBOOST to multiclass problems. Our algorithm, SFBOOST.MC, has several interesting features. First, it builds a single multiclass SF. Therefore, it does not require classifier combination tricks such as those for linear separators [9]. Second, it does not require anymore the observations to be binary described, as usual for SFs. Third, under hypotheses closely matching weak learning’s [9], the error of the optimal SF vanishes rapidly. Fourth and most importantly, the algorithm turns out to be the multiclass generalization of the algorithms of [5]. Therefore, it enjoys desirable properties such as strong noise tolerance. Section 2 details some notations. Section 3 presents SFBOOST.MC. Section 4 presents some theoretical properties of the algorithm, and Section 5 details some experiments.

2 Notations and preliminaries

Due to the space constraints, we shall assume basic knowledge of boosting algorithms, and in particular of their main representative: ADBOOST[9]. We let \(LS = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}\) denote a set of \(|LS| = m\) training examples, where \(|\cdot|\) denotes the cardinal. Each observation (or instance) \(x_i\) belongs to a domain \(X\), and is described using \(n\) variables. Each \(y_i\) is a label (or class), belonging to a set \(\mathcal{Y}\) with \(|\mathcal{Y}| = c < \infty\). Without loss of generality, we shall number the classes into the set \(\{0, 1, \ldots, c - 1\}\). When \(c = 2\), we follow the convention \(\mathcal{Y} = \{-1, +1\}\) for classes 0 (the "negative" class) and 1 (the "positive" class). We consider that a SF is a function \(H : \{0, 1\}^n \rightarrow \mathcal{Y}\) which is invariant under permutation of the input bits. When \(c = 2\), we can replace without loss of
some observation which falls into the bucket at time \( j \),
\( I = 0 \) for which \( x = 0 \) as in [7], and consider that the sign of \( \alpha \) returns a
\( \rightarrow \) denotes \( \alpha \) of concept representation
\( \sum \). For \( \alpha \) in its
\( \{ x, y \} \) makes a partition of \( X \) is based on a trans-
dependent notation \( \alpha_{t,x}^{y,y'} \), with \( j = \sum_{i=1} h_i(x) \) or \( j = 0, \)
1 intermediate hypotheses \( (h_1) \)
\( \Rightarrow 1 \) new description variable
\( \Rightarrow 2 \) buckets
2 intermediate hypotheses \( (h_1, h_2) \)
\( \Rightarrow 2 \) new description variables
\( \Rightarrow 3 \) buckets
more intermediate hypotheses \( (h_1, h_2, \ldots) \)

Figure 1. A schematic representation of the way \( H_T(.) \) is built, when \( c = 3 \). As the number of intermediate hypotheses increases, the number of buckets also increases, and thus the chances of getting better partitions of \( LS \). On the right, functions \( H_t(.) \) are shown for \( t = 0, 1, 2 \), with their associated bucket vectors as square colors (see text for details).

3 Multiclass induction of SF

\forall t = 0, 1, \ldots, T, \forall j = 0, 1, \ldots, c, we associate to the \( j \)th bucket of \( H_t \) a \( c \times c \) matrix \( A_{t,j} \) whose entries are noted \( a_{t,j}^{y,y'} \) (\( y \neq y' \)). Informally, coefficient \( a_{t,j}^{y,y'} \) in \( A_{t,j} \) represents the confidence in predicting class \( y \) against class \( y' \). The class output by \( H_T \) in its \( j \)th bucket, \( \bar{\nu}_T[j] \), is its majority class:

\[ \bar{\nu}_T[j] = \arg \max_{y \neq y'} \max_{y,y'} a_{t,j}^{y,y'} \quad . \]

Equivalently, \( \bar{\nu}_T[j] \) is the row number \( (y) \) in \( A_{t,j} \) for which \( \alpha_{t,j}^{y,y'} \geq \alpha_{t,j}^{y,y'} \) \( y \neq y' \). In case of ties, it is one of the rows for which this holds.

The induction of SFs for arbitrary \( c \geq 2 \) is based on a transformation of \( LS \). Each \( (x, y) \in LS \) gives rise to \( c - 1 \) new examples \( \{(x, y, y')\}, \forall y' \neq y \). We let \( LS' \) denote this new set of examples (when \( c = 2 \), we can consider that \( LS' = LS \)). This transformation is similar to one of [9]. The initial distribution of the examples of \( LS' \), \( D_0 \), is uniform: \( D_0((x, y, y')) = 1/(m(c - 1)) \). In the following, we shall name without ambiguity a bucket obtained through the run of SFBOOST.MC either by some couple \((t, j)\), with \( t = 1, 2, \ldots, T \) and \( j = 0, 1, \ldots, c \), or by some couple \((t, x)\), with \( t \) some observation which falls into the bucket at time \( t \). Thus, we may also replace \( \alpha_{t,j}^{y,y'} \) by an observation-dependent notation \( \alpha_{t,x}^{y,y'} \), with \( j = \sum_{i=1} h_i(x) \) or \( j = 0, \)

generality \( \mathcal{Y} \) by \( \mathcal{R} \) as in [7], and consider that the sign of the output gives the label. Suppose we have \( T \) intermediate hypotheses, \( h_1, h_2, \ldots, h_T \), each element of some class of concept representation \( \mathcal{H} \), with the sole restriction that each \( h_t \) \( (t = 1, 2, \ldots, T) \) maps \( \mathcal{X} \) onto \{0, 1\} (regardless of \( c \)). This is an important difference with classical boosting algorithms such as ADAUST, for which each \( h_t \) returns a class in \( \mathcal{Y} \) [9]. Building a SF \( H_T \) using this intermediate set of hypotheses boils down to building a SF over the transformed set of \( (x_1', y_1), (x_2', y_2), \ldots, (x_m', y_m) \}, \) with \( x_i' = \sum_{j=1}^T h_j(x_i) \). \( H_T \) makes a partition of \( \mathcal{X} \) into what we call buckets, the \( j \)th bucket \( (j = 0, 1, \ldots, T) \) receiving the examples \( (x_i, y_i) \) for which \( \sum_{j=1}^T h_j(x_i) = j \). The output of \( H_T \) can be represented by a \( T + 1 \) dimension bucket vector \( \bar{\nu}_T \), such that \( H_T(x) = \bar{\nu}_T \left[ \sum_{j=1}^T h_j(x) \right] \in \mathcal{Y} \). Note that even when we have no intermediate hypothesis, a SF \( H_0 \) can already be built: its 1-dimension bucket vector returns the majority class of \( LS \).

SFBOOST.MC repeatedly requests intermediate hypotheses to build the final SF. \( T \) intermediate SFs are built \( (H_0, H_1, \ldots, H_{T-1}) \), with each example following a path through their buckets. The arrows of Figure 1 depict possible paths. If the \( h_t \)s satisfy good properties, then one might rapidly obtain, as \( t \) increases, partitions of \( LS \) with each bucket having an increasing gap between the majority class and all others (see e.g. \( H_3 \) in Figure 1). Before drilling down these properties, we formally present SFBOOST.MC.
for $A_{0,0}$. Finally, we let $\text{Interm}_\text{hyp}(L'S', D_t)$ denote the intermediate learner, which takes as input $L'S'$, its distribution at time $t$, $D_t$, and returns an hypothesis $h_t \in \mathcal{H}$. $\text{SFBOOST.MC}$ is shown below (Algorithm $\text{SFBOOST.MC}$). When $c = 2$, it is equivalent to [7]. Thus, $\text{SFBOOST.MC}$ is a strict generalization of [7]'s $\text{SFBOOST}$.

Algorithm 1: $\text{SFBOOST.MC}(L'S')$

| Input: $L'S' = \{(x, y, y'): (x, y) \in L{S'} \wedge y' \neq y\) |
| for $i = 1$ to $m$ do $D_0((x, y, y')) = 1/(m(c - 1))$; |
| for $y = 0$ to $c - 1$ do |
| for $y' = 0$ to $c - 1$ do $\alpha_{0,0}^{y,y'} = \frac{1}{2} \ln(D_0^{y,y'}/D_0^{y,0})$; |
| $Z_0 = \sum_{(x, y, y') \in L'S'} D_0((x, y, y')) \exp(-\alpha_{0,0}^{y,y'})$; |
| for $(x, y, y') \in L'S'$ do |
| $D_1((x, y, y')) = (D_0((x, y, y')) \exp(-\alpha_{0,0}^{y,y'})/Z_0$; |
| for $t = 1$ to $T$ do |
| $h_t = \text{Interm}_\text{hyp}(L'S', D_t)$; |
| for $j = 0$ to $t$ do |
| $LS_{t,j} = \{(x, y, y') \in L'S': \sum_{i=1}^{t} h_i(x) = j\}$; |
| $\alpha_{y,j} = \frac{1}{2} \ln \sum_{LS_{t,j}} D_t((x, y, y')) \exp(\alpha_{t-1,j}^{y,y'})$; |
| $Z_t = \sum_{LS_{t,j}} D_t((x, y, y')) \exp(-\alpha_{t-1,j}^{y,y'}) + \alpha_{t-1,j}^{y,y'}$; |
| for $(x, y, y') \in LS_{t,j}$ do |
| $D_{t+1}((x, y, y')) = D_t((x, y, y')) \exp(-\alpha_{t-1,j}^{y,y'})$; |
| Output: $H_T(x) = \arg \max_{y} \max_{y'} \alpha_{t,j}^{y,y'} Z_t$ |

4 Properties of $\text{SFBOOST.MC}$

We first prove that $\text{SFBOOST.MC}$ bears similarities with boosting’s most representative algorithm: AdaBoost. The following Theorem is quite similar (in the multiclass case) to Theorems 1, 4, 5, 6 in [9]. For any predicate $\pi$, let $[\pi]$ be 1 if $\pi$ holds, and 0 otherwise.

Theorem 1 The error of $H_T$ on $L{S'}$ satisfies:

$$(1/m)\{|(x, y) \in L{S'}: H_T(x) \neq y\} \leq (c - 1) \prod_{t=0}^{T} Z_t.$$

(proof omitted due to the lack of space) Theorem 1 has the same consequence as Theorems (1, 4, 5, 6) in [9]; it brings a criterion that the intermediate learner should minimize: the normalization coefficient $Z_t$. The expression of $A_{t,j}$’s entries in eq. (2) is obtained using exactly the same method as [9].

Finally, we can study the malicious noise tolerance of $\text{SFBOOST.MC}$. [5] have studied the learnability of concepts when data can be corrupted by errors from whom absolutely no assumption can be made. Their “malicious noise” model takes place in which an adversary manipulates any example with probability $\beta$, to return something from which nothing can be assumed. This adversary has unbounded computational resources, knows everything about the task, and knows the internal state of the learner. Informally, [5] show that the highest amount of noise which can be tolerated by any learning algorithm is no more (up to a constant factor) than the desired error. They also show that the SF admits an algorithm which does not only tolerate this optimal bound, but also achieves minimal sample complexity. It turns out that, at each time $t$, the SF $\text{SFBOOST.MC}$ builds the same as the one which would be chosen in Theorem 11 of [5].

5 Experiments

As $\text{SFBOOST.MC}$ proceeds by repeatedly splitting the training sample into subsamples, there may be some problems to compute the matrices $A_{t,j}$ (eq. 2) whenever the weight of one class approaches zero, or equals zero in a bucket, which in turn would severely bias the update of distribution $D_t$. To avoid such situations, we have chosen to follow experimentally the setup proved by [9], which boils down to replacing the expression of the coefficients $\alpha_{t,j}^{y,y'}$ by the following one $\alpha_{t,j}^{y,y'} = \frac{1}{2} \ln(D_t^{y,y'}/D_t^{y,0})$.

Figure 2. Class frequencies for the buckets of $L{S'}$ and $H_t$ ($t = 0, 1, 3, 5$). Horizontal axis denotes bucket numbers. For all domains, we have $l = 2, r = 5$. 

iris ($c = 3$) horse ($c = 2$) wine ($c = 3$)
in some bucket. Note that on domains with large number of classes in each bucket, when no intermediate hypothesis is brought, all examples share the same empty description, and therefore the sole SF which can be built contains a single bucket. In this bucket, the proportions of the classes are the observed proportions of examples in $LS$. As $t$ increases, we add buckets one after the others, and $LS$ becomes partitioned into an increasing number of buckets. Hopefully, we expect to get buckets with increasing gaps between their majority class and the others. This could be observed e.g. by coefficients $\alpha_{t,j}^{y,y'}$ of increasing magnitudes on the row of the majority class. Experimentally, we have plotted the curves representing the relative proportions of classes in each bucket as a function of the bucket number, for some of the first six SFs built on three UCI domains [2] (these are $H_0, H_1, H_2, H_3$). Figure 2 presents the results obtained. For each domain, the plots should be read from the top ($LS$) to the bottom plot ($t = 5$). We observe that SFBoost.MC tends quite rapidly to separate the classes, with one majority class becoming more and more prominent in each bucket. Even more, we have remarked that when $t = (c - 1)$, i.e. when the number of buckets equals the number of classes, each bucket has a different majority class. This seems to account for an early transition to good accuracy (see below), as each class achieves majority in some bucket. Note that on domains with large number of classes ($c > 20$), this does not hold anymore. This might be due to the simplicity of the intermediate learner’s formulas, with monomials unable to separate early and efficiently the classes for large $c$. Finally, the horse domain curves are very smooth, and there is only one region in the SF where the two curves cross each other (buckets 2/3 at $t = 5$). This phenomenon, which we have observed for many other datasets, might be the starting point of further work on prototype selection, to measure the influence on generalization of discarding those examples located on the frontier buckets.

Finally, we have tested the performance of SFBoost.MC against AdaBoost [9] and C4.5 [8]. Each algorithm was tested on a set of 25 domains, most of which come from the UCI repository of ML/DM database [2]. The error is evaluated by averaging over a ten-fold stratified cross validation procedure. For the sake of comparison, we have chosen for the weak/intermediate learners (for SFBoost.MC and AdaBoost) a simple class of concept representations: monomials with a maximal number of literals $\leq l$ for some $l > 0$. Each algorithm is ran with a fixed value for $l$, and requests a number of rules equal to $r$, for some $r > 0$. As suggested by theory, the weak learners are designed to optimize respectively $Z_t$ (ineq. (3) for SFBoost.MC ) and the $Z$ of AdaBoost (section 3 in [9] for AdaBoost ). The weak/intermediate learners are also stepwise greedy optimization procedures for their corresponding $Z$ criterion, building each monomial from scratch. Figure 3 summarizes the results obtained against AdaBoost and C4.5 over each of the 25 datasets, for couples of values $(l, r) \in \{(1, 10), (2, 10)\}$. They depict the ability of SFBoost.MC to improve on the results of AdaBoost and C4.5 on the majority (18) of domains. A simple sign test confirms this behavior, as it reveals a $p = 0.0216$ threshold probability to reject the hypothesis of an identical behavior on the 25 datasets of SFBoost.MC vs AdaBoost, or SFBoost.MC vs C4.5.

References