When Analytic Calculus Cracks AdaBoost Code

Jean-Marc Brossier, Olivier Lafitte, Lenny Réthoré

WHEN ANALYTIC CALCULUS CRACKS ADABOOST CODE

JEAN-MARC BROSSIER[§], OLIVIER LAFITTE[†], AND LENNY RÉTHORÉ[§]

Abstract. The principle of boosting in supervised learning involves combining multiple weak classifiers to obtain a stronger classifier. ADABOOST has the reputation to be a perfect example of this approach. It has been shown in [3] that ADABOOST is not truly an optimization algorithm.

This paper shows that ADABOOST is an algorithm in name only, as the resulting combination of weak classifiers can be explicitly calculated using a truth table.

This study is carried out by considering a problem with two classes and is illustrated by the particular case of three binary classifiers and presents results in comparison with those from the implementation of ADABOOST algorithm of the Python library scikit-learn.

1. Introduction

Consider a dataset $\mathcal{S} = \{(x_i, y_i)\}_{i=1..n} \subset \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X} = \mathbb{R}^d$ is a set of characteristics and $\mathcal{Y} = \{-1, +1\}$ a set of labels for two classes.

We want to classify these examples so that the obtained classifier $h : \mathcal{X} \to \mathcal{Y}$ matches each data point x_i with its label y_i with the fewest errors. Thus, we want to find a classifier h which is equal to y_i as many times as possible.

This can be done by studying a convexified version of an empirical risk over a given convex set of classifiers $\mathcal{H}$ as in [1].

Considering $\mathbf{G} = (G_1, G_2, G_3)$ three weak binary classifiers and their weights $\boldsymbol{\beta} = (\beta_1, \beta_2, \beta_3) \in \mathbb{R}^3$ in the resulting classifier $h = \operatorname{sign}(\boldsymbol{\beta} \cdot \mathbf{G})$, this convexified empirical risk, using the convex function $\exp(-x)$, is:

$$\mathcal{R}(\boldsymbol{\beta}, \mathcal{S}) = \frac{1}{n} \sum_{i=1}^{n} \exp\left(-y_i \boldsymbol{\beta} \cdot \mathbf{G}(x_i)\right).$$

We rewrite this risk by considering an approach based on truth tables, as in [3].

Let p be the number of weak binary classifiers (here, we mostly deal with $p = 3$). Given that $(y_i, G_k(x_i)) \in \mathcal{Y}^2$, for any $k \in [\![1; p]\!]$ and $i \in [\![1; n]\!]$, the product $y_i G_k(x_i)$ is either equal to $+1$ if y_i and $G_k(x_i)$ are of the same sign, that is $G_k(x_i)$ is true, or equal to -1 if y_i and $G_k(x_i)$ are not of the same sign, that is $G_k(x_i)$ is false. For a list of p weak classifiers, this leads to 2^p possible combinations of the p classifiers which yields a partition of 2^p subsets of $\{x_i, i = 1..n\}$. We thus create a truth table of p rows for all classifiers and 2^p columns for all configurations, encompassing

[§]CNRS, Univ. Grenoble Alpes, Grenoble-INP, GIPSA-lab, Grenoble, France

[†]Université Sorbonne Paris Nord, LAGA, UMR 7539. IRL CNRS-CRM 3457. Université de Montréal. Canada

all values of $y_i G_k(x_i)$. For example, we have the following truth table for $p = 3$ (labelling the sign of $y_i G_k(x_i)$ by G_k for simplicity):

	n_0	m_0	n_1	m_1	n_2	m_2	n_3	m_3
G_1	-1	1	1	-1	-1	1	-1	1
G_2	-1	1	-1	1	1	-1	-1	1
G_3	-1	1	-1	1	-1	1	1	-1
$\boldsymbol{\beta}^T\mathbf{G}$	$-X_0$	X_0	$-X_1$	X_1	$-X_2$	X_2	$-X_3$	X_3

The coefficients n_j and m_j, for $j \in [\![0; 2^{3-1} - 1]\!]$, count the number of occurrences of the corresponding configurations in $\mathcal{S}$, with $\sum_j (n_j + m_j) = n$. For example, m_1 counts the number of elements misclassified by G_1, but correctly classified by G_2 and G_3. Each of the $2^3 = 8$ configurations, or columns, is associated to one of the 8 quantities $\pm X_0, \cdots, \pm X_3$ defined by $X_1 = -\beta_1 + \beta_2 + \beta_3$, $X_2 = \beta_1 - \beta_2 + \beta_3$, $X_3 = \beta_1 + \beta_2 - \beta_3$ and $X_0 = \beta_1 + \beta_2 + \beta_3 = X_1 + X_2 + X_3$.

The risk rewrites, using this logic approach:

$$\mathcal{R}(\boldsymbol{\beta}, \mathcal{S}) = \frac{1}{n} \sum_{j=0}^{2^{3-1}-1} \left(n_j e^{X_j} + m_j e^{-X_j} \right). \tag{1}$$

According to many authors, ADABOOST is an algorithm returning the point of minimum of this convexified empirical risk. However, it has been demonstrated in [3] that ADABOOST does not minimize (1) and therefore is not an optimization algorithm. In this paper, we will show that it is not an algorithm either in the sense that, the truth table being obtained, we can determine upstream the weight β_k of each weak estimator G_k in the resulting classifier h with very simple calculations and without the help of such an algorithm[1].

2. Calculation of the estimators weights for the general case

The traditional ADABOOST program as proposed initially by [4], is a procedure in which, over the course of iterations, the weight w_i of each example x_i is updated by calling a predefined *weaklearner*. The weight is modified according to whether the example is correctly classified or not by the new *weaklearner*: if the example is misclassified by the *weaklearner*, its weight w_i is increased.

Note that, in algorithm 1, we use the notation α_k for the estimator weights to follow [5]. One notices that $\alpha_k = 2\beta_k$. The weight α_k of each *weaklearner* is computed at each iteration k, based on the error ϵ_k it generates. This error ϵ_k is weighted by the weights w_i of the examples, which are also updated at each iteration. Note that, in certain descriptions of this algorithm, one exits the loop when $\epsilon_k \geq \frac{1}{2}$. However, the algorithm as it is coded in scikit-learn escapes this issue and keeps going, discarding in the weighted configuration the classifiers whose error is greater than $\frac{1}{2}$.

Although it could appear to be challenging to extract the values of each weight α_k, using truth tables without taking into account any examples weighting w_i returns the analytic values of each weight α_k as follows.

[1]As implemented in the Python library scikit-learn.

Algorithm 1 ADABOOST

Input: Dataset of n examples $S = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}, i \in [\![1, n]\!]\}$
Integer p specifying the number of iterations

Ensure: $w_i = \frac{1}{n}$ for $i \in [\![1, n]\!]$ the weight vector of each example

for $k \in [\![1, p]\!]$ **do**
Fit a classifier G_k to the dataset $\mathcal{S}$ using weights w_i
$\epsilon_k \leftarrow \Big(\sum_{i=1}^{n} w_i 1_{G_k(x_i) y_i < 0}\Big) / \Big(\sum_{i=1}^{n} w_i\Big)$
$\alpha_k \leftarrow \ln\big((1 - \epsilon_k)/(\epsilon_k)\big)$
for $i \in [\![1, n]\!]$ **do**
$w_i \leftarrow w_i \exp\Big(\alpha_k 1_{G_k(x_i) y_i < 0}\Big)$
end for
end for

Output: $h(x_i) = \operatorname{sign}\Big(\sum_{k=1}^{p} \alpha_k G_k(x_i)\Big)$

The ADABOOST *analytic formula* allowing to obtain the weights $\boldsymbol{\beta}$ explicitly, is the aim of this paper.

We proceed in an incremental way and we compute the weight β_k at step $k \leq p$ using only the truth tables and the weights $\boldsymbol{\beta}^{(k-1)} = (\beta_1, \ldots, \beta_{k-1})$ obtained in the previous steps. These weights form a first-order recurrent sequence on β_k. We set $\tau_q = e^{\beta_q}$ for $q \in [\![1, k-1]\!]$. The quantities $\tau_1, \ldots, \tau_{k-1}$ allow to go from step $k-1$ to the next step k and is a necessary adjunction to the truth tables.

The final truth table (at step p), seen in the introduction, is constructed in an incremental way from the tables of lower order: at step k, the weight β_k of the classifier G_k is computed using the truth table at step $k-1$ and the action of the *weaklearner* G_k. Hence, we consider a truth table for each step k.

For example, for $p = 3$, we have 3 truth tables:

	c_2	c_3
G_1	-1	1

	c_4	c_5	c_6	c_7
G_1	-1		1	
G_2	-1	1	-1	1

	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
G_1	-1				1			
G_2	-1		1		-1		1	
G_3	-1	1	-1	1	-1	1	-1	1
	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
	n_0	n_3	n_2	m_1	n_1	m_2	m_3	m_0

The coefficients c_l, as the coefficients n_j and m_j seen before, count the number of occurrences (cardinal) of each configuration described by a subset $\mathcal{S}_l$ of $\mathcal{S} = \mathcal{S}_1$. We

have thus $c_1 = n$ and we can match each $c_8, \dots, c_{15}$ to a corresponding $n_0, \dots, n_3$ or $m_0, \dots, m_3$ for the case $p = 3$. The weights β_k depend only on the coefficients c_l associated with the truth tables at step k.

We thus define a tree structure of disjoint subsets of $\mathcal{S} = \mathcal{S}_1$ such that, $\forall k \leq p$, $\mathcal{S} = \bigsqcup_{j=2^k}^{2^{k+1}-1} \mathcal{S}_j$ and $n = \sum_{j=2^k}^{2^{k+1}-1} c_j$.

This corresponds to the Sosa-Stradonitz numeration in [7] of a genealogical tree and the recurrence relation $c_j = c_{2j} + c_{2j+1}$ holds true at any step k for any $j \in [\![2^{k-1}; 2^k - 1]\!]$.

For each c_j, we construct $\boldsymbol{\epsilon}(j) \in \mathcal{Y}^{k-1}$ which retraces the genealogy of c_j thanks to the $(k-1)^{th}$ truth table. For instance, we have $\boldsymbol{\epsilon}(5) = (-1, 1)$ and $\boldsymbol{\epsilon}(13) = (1, -1, 1)$.

The risk at step $k-1$ is thus $\sum_{j=2^{k-1}}^{2^k-1} c_j e^{-\boldsymbol{\epsilon}(j).\boldsymbol{\beta}^{(k-1)}}$. Applying the *weaklearner* G_k at step k, the convexified risk becomes $\sum_{j=2^{k-1}}^{2^k-1} \left(c_{2j} e^{\beta} + c_{2j+1} e^{-\beta}\right) e^{-\boldsymbol{\epsilon}(j).\boldsymbol{\beta}^{(k-1)}}$. Let $\widetilde{C}_{2j} = \frac{c_{2j}}{c_j} C_j$ and $\widetilde{C}_{2j+1} = \frac{c_{2j+1}}{c_j} C_j$, with the weights $C_j = c_j e^{-\boldsymbol{\epsilon}(j).\boldsymbol{\beta}^{(k-1)}}$ yielding $C_{2j} = \widetilde{C}_{2j} \tau_k$ and $C_{2j+1} = \frac{\widetilde{C}_{2j}}{\tau_k}$.

Hence, we rewrite the convexified risk at step k as:

$$\mathcal{R}_k(\beta, \mathcal{S}) = \sum_{j=2^{k-1}}^{2^k-1} \widetilde{C}_{2j} e^{\beta} + \widetilde{C}_{2j+1} e^{-\beta}. \tag{2}$$

We have $\mathcal{R}_k(\beta, \mathcal{S}) = \mathcal{R}\left(\left(\boldsymbol{\beta}^{(k-1)}, \beta, 0, \dots, 0\right), \mathcal{S}\right)$ and this function achieves its minimum at point $\beta = \beta_k$.

The weight β_k, which minimizes $\mathcal{R}_k(\beta, \mathcal{S})$, is thus:

$$\beta_k = \frac{1}{2} \ln\left(\frac{b_k}{a_k}\right), \tag{3}$$

where $a_k = \sum_{j=2^{k-1}}^{2^k-1} \widetilde{C}_{2j}$ and $b_k = \sum_{j=2^{k-1}}^{2^k-1} \widetilde{C}_{2j+1}$, thanks to $\mathcal{R}_k(\beta, \mathcal{S}) = a_k e^{\beta} + b_k e^{-\beta}$.

Hence, we repeat the process until we obtain the whole set of weights $\boldsymbol{\beta} = \boldsymbol{\beta}^{(p)}$. Note that β_1 is calculated using $\widetilde{C}_2 = c_2$ and $\widetilde{C}_3 = c_3$ from which we deduce all the subsequent β_k. We recover analytically what ADABOOST computes: ADABOOST is nothing more than extensive calculations of the weights $\boldsymbol{\beta}^{(p)}$. We recall that $\boldsymbol{\beta}^{(p)}$ is not the point of minimum of the convexified risk and doesn't verify equation (4). On the other side, (4) reverts, for $p = 3$, to solve a single equation (and for $p > 3$ to a system of 2^p equations with $2^p - p$ constraints).

An exact minimization formula introduced in [3], is obtained through the resolution of a scalar equation: the weights $\boldsymbol{\beta}$ which is the exact point of minimum of the convexified cost function corresponds to solving the equation:

$$\prod_{j=0}^{3} \left(\frac{T_0}{2} + \sqrt{\left(\frac{T_0}{2}\right)^2 + m_j n_j}\right) - m_0 n_1 n_2 n_3 = 0, \tag{4}$$

which leads to $n_0 e^{X_0} = \frac{T_0}{2} + \sqrt{\left(\frac{T_0}{2}\right)^2 + m_0 n_0}$ and similar formulae for X_1, X_2 and X_3, hence giving the optimal $\boldsymbol{\beta}$.

3. The particular case of three weak learners

We calculate the weights $\boldsymbol{\beta}^{ada}$ in the case of $p = 3$. To compute the first weight β_1^{ada}, we simply apply the formula we have derived with the first truth table.

$$\beta_1^{ada} = \ln\left(\sqrt{\frac{\widetilde{C}_3}{\widetilde{C}_2}}\right) = \ln\left(\sqrt{\frac{c_3}{c_2}}\right).$$

We can now set the first factor $\tau_1 = e^{\beta_1^{ada}} = \sqrt{\frac{c_3}{c_2}}$.

We have to compute $\widetilde{C}_{2j}$ and $\widetilde{C}_{2j+1}$, $j = 2, 3$, at step $k = 2$ using τ_1 and the coefficients of the second truth table c_{2j} and c_{2j+1}, $j = 2, 3$, thanks to the relation we set: $\forall j \in [\![2^{k-1}; 2^k - 1]\!]$, $\widetilde{C}_{2j} = \frac{c_{2j}}{c_j} C_j$ and $\widetilde{C}_{2j+1} = \frac{c_{2j+1}}{c_j} C_j$, with $C_j = c_j e^{-\boldsymbol{\epsilon}(j)\cdot\boldsymbol{\beta}^{(k-1)}}$.

We have : $\widetilde{C}_4 = c_4\tau_1$, $\widetilde{C}_5 = c_5\tau_1$, $\widetilde{C}_6 = \frac{c_6}{\tau_1}$ and $\widetilde{C}_7 = \frac{c_7}{\tau_1}$.

Hence, we can directly deduce β_2^{ada} as follows:

$$\beta_2^{ada} = \ln\left(\sqrt{\frac{\widetilde{C}_5 + \widetilde{C}_7}{\widetilde{C}_4 + \widetilde{C}_6}}\right).$$

In the same way, we set $\tau_2 = e^{\beta_2^{ada}}$ and we compute the values $\widetilde{C}_{2j}$ and $\widetilde{C}_{2j+1}$ at step $k = 3$ thanks to τ_1, τ_2 and the coefficients of the third truth table c_{2j} and c_{2j+1}, where we have in this case: $\tau_2 = \sqrt{\frac{\widetilde{C}_5+\widetilde{C}_7}{\widetilde{C}_4+\widetilde{C}_6}}$.

So: $\widetilde{C}_8 = c_8\tau_1\tau_2$, $\widetilde{C}_9 = c_9\tau_1\tau_2$, $\widetilde{C}_{10} = \frac{c_{10}\tau_1}{\tau_2}$, $\widetilde{C}_{11} = \frac{c_{11}\tau_1}{\tau_2}$, $\widetilde{C}_{12} = \frac{c_{12}\tau_2}{\tau_1}$, $\widetilde{C}_{13} = \frac{c_{13}\tau_2}{\tau_1}$, $\widetilde{C}_{14} = \frac{c_{14}}{\tau_1\tau_2}$ and $\widetilde{C}_{15} = \frac{c_{15}}{\tau_1\tau_2}$.

Hence, we can finally compute the last classifier weight:

$$\beta_3^{ada} = \ln\left(\sqrt{\frac{\widetilde{C}_9 + \widetilde{C}_{11} + \widetilde{C}_{13} + \widetilde{C}_{15}}{\widetilde{C}_8 + \widetilde{C}_{10} + \widetilde{C}_{12} + \widetilde{C}_{14}}}\right).$$

We have thus found all the weights of the weak estimators computed by ADABOOST using only truth tables and without taking into account any example weight directly.

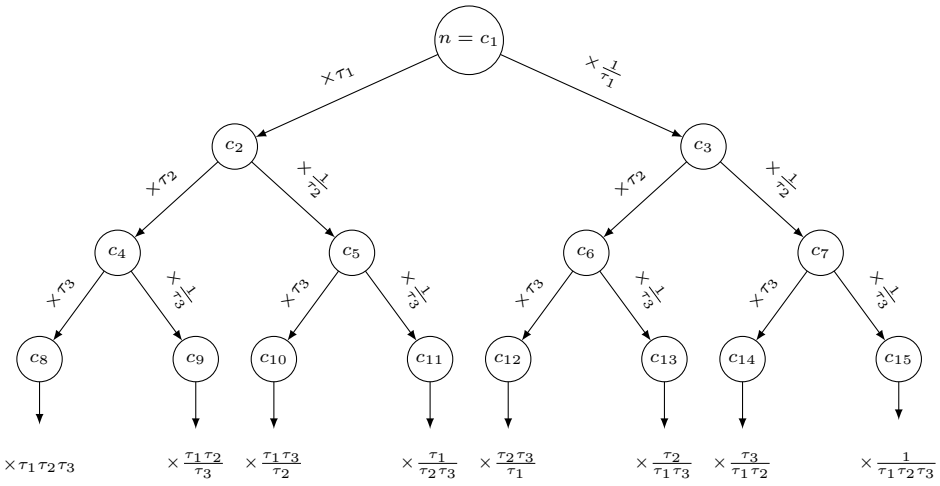
Moreover, instead of introducing the $\widetilde{C}_j$ and C_j, we can express each β_k^{ada} only in terms of the original coefficients c_j of the truth tables. With $\tau_1 = \sqrt{\frac{c_3}{c_2}}$ and

$\tau_2 = \sqrt{\frac{c_5\tau_1+\frac{c_7}{\tau_1}}{c_4\tau_1+\frac{c_6}{\tau_1}}}$, one indeed has:

$$
(5) \qquad \begin{cases} \beta_1^{ada} = \ln\left(\sqrt{\frac{c_3}{c_2}}\right), \\ \beta_2^{ada} = \ln\left(\sqrt{\frac{c_5\tau_1+\frac{c_7}{\tau_1}}{c_4\tau_1+\frac{c_6}{\tau_1}}}\right) \\ \beta_3^{ada} = \ln\left(\sqrt{\frac{c_9\tau_1\tau_2+\frac{c_{11}\tau_1}{\tau_2}+\frac{c_{13}\tau_2}{\tau_1}+\frac{c_{15}}{\tau_1\tau_2}}{c_8\tau_1\tau_2+\frac{c_{10}\tau_1}{\tau_2}+\frac{c_{12}\tau_2}{\tau_1}+\frac{c_{14}}{\tau_1\tau_2}}}\right). \end{cases}
$$

This allows to get the value of $X_0 = \beta_1^{ada} + \beta_2^{ada} + \beta_3^{ada}$, expression which can be used to test (4).

This is summarized by the following tree structure:



Moreover, this tree structure shows the case where τ_3 is used to weighting the exemples before applying G_4 (for $p > 3$).

Therefore, we have successfully calculated the values of all the weights $\boldsymbol{\beta}^{ada} = (\beta_1, \beta_2, \beta_3) \in \mathbb{R}^3$, and we can directly deduce the resulting classifier of the ADABOOST algorithm without even running it.

Moreover, note that there are obvious relations between the coefficients n_j and m_j and the coefficients c_l. For instance, with $p = 3$, we have: $c_2 = n_0 + m_1 + n_2 + n_3$, $c_3 = m_0 + n_1 + m_2 + m_3$, $c_4 = n_0 + n_3$, $c_5 = m_1 + n_2$, $c_6 = n_1 + m_2$, $c_7 = m_0 + m_3$, $c_8 = n_0$, $c_9 = n_3$, ...

This means that the set of weights $\boldsymbol{\beta}^{ada}$ returned by ADABOOST can be deduced over the course of iterations with the c_l as well as with the n_j and m_j.

The weights $\boldsymbol{\beta}^{ada}$ obtained algebraically rigorously here lead to a highly powerful and very cheap (in terms of computation time) method (see below). The structuration of the data provided by the truth tables makes ADABOOST in scikit-learn superfluous: the only sufficient information needed that ADABOOST can provide is the specific

weaklearner it calls. The choice of the *weaklearner* in scikit-learn is thus really crucial.

4. Simulations Results for Three Classifiers

Based on the previous results, we analytically calculate the weights $\boldsymbol{\beta}$ that the ADABOOST algorithm in scikit-learn. It is important to note that this scikit-learn implementation is based on the algorithm by [6], which is equivalent to the algorithm by [4] in the binary classification case we are dealing with. We also recall that ADABOOST as implemented in scikit-learn uses $\alpha_k = 2\beta_k$. To construct a set $\mathbf{G} = (G_1, G_2, G_3)$ of weak classifiers and derive a truth table, we run the ADABOOST algorithm from scikit-learn for $p = 3$ iterations on a random dataset of 1000 examples in $\mathbb{R}^2$ with two classes distributed according to a Gaussian distribution. The misclassified data are highlighted in cyan in Fig. 1:

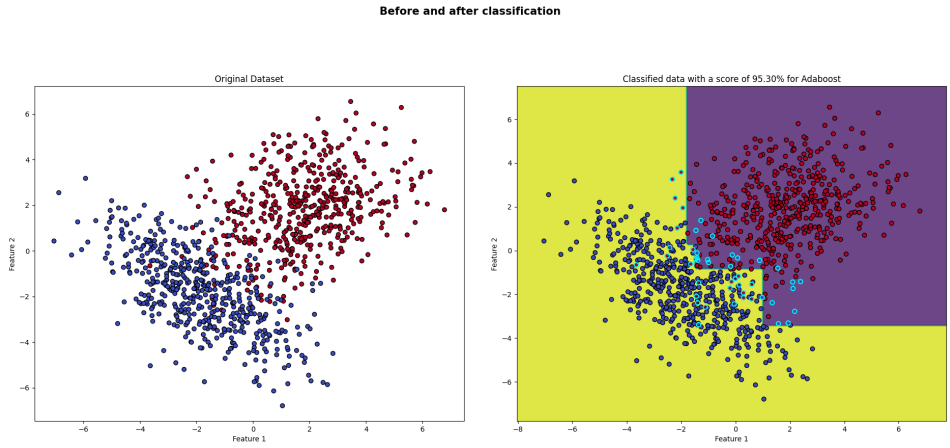


Figure 1. *An example of classification performed by* ADABOOST *from Scikit-Learn for* $p = 3$ *CART decision trees [2] with* $T = 4$ *terminal points*

This allows us to obtain both the set of classifiers $\mathbf{G}$ and the values of the computed weights $\boldsymbol{\beta}$ in order to compare them with our calculated values. With the set $\mathbf{G}$ of classifiers, we can finally construct the truth table, for example:

$p=3$	n_0	m_0	n_1	m_1	n_2	m_2	n_3	m_3
	4	767	9	42	18	44	16	100
G_1	-1	1	1	-1	-1	1	-1	1
G_2	-1	1	-1	1	1	-1	-1	1
G_3	-1	1	-1	1	-1	1	1	-1

The values of n_j and m_j have been calculated for $p = 3$ decision trees (default weak classifier used by scikit-learn) with $T = 4$ terminal points as described in [2].

The weights of the classifiers returned by ADABOOST are $\boldsymbol{\beta}^{ada} = (1.221, 0.852, 0.706)$. The approach using the truth table returns the same $\boldsymbol{\beta}$, up to a mean absolute error of 2.96×10^{-16}, in a reduced amount of time (33µs against 5ms for ADABOOST). We obtain the same resulting classifier.

There is, however, a surprising feature of the procedure ADABOOST in scikit-learn: when the *weaklearner* returns a weighted error greater than $\frac{1}{2}$, the weight β^{ada} is computed to update as usual the examples weights w_i, but this *weaklearner* is not used in the subsequent classifiers list, *i.e.* for the final classification, as if β^{ada} was then set to 0. However, this has no consequence on our results since the algorithm still computes a value of β^{ada} to update the weights of the examples before setting it to 0. Therefore, our approach retrieves the value of β^{ada} before this forced setting to 0.

This example on a single dataset $\mathcal{S}$ illustrates the beauty of these calculations. To be more exhaustive, we calculate this mean absolute error for 2000 random datasets in the same conditions (1000 examples in $\mathbb{R}^2$, 2 classes, Gaussian distribution, 3 weak classifiers). The total mean absolute error over these 2000 random datasets is $2,99 \times 10^{-16}$. Our approach thus computes exactly what ADABOOST returns, through a very stable calculus. This stability does not depend on the distribution of the examples, on the number of examples or on the *weaklearners* considered. Note, however, that the procedure of weighting the misclassified examples changes the probability law of the dataset $\mathcal{S}$.

As plugging T_0 deduced from $\boldsymbol{\beta}^{ada}$ in (4) does not return 0, we show that $\boldsymbol{\beta}^{ada}$ is not the point of minimum of the convexified cost function and thus that ADABOOST is not a minimization algorithm. On the contrary, the point of minimum calculated in [3] satisfies (4).

5. Conclusion

Our logical approach recovers the resulting classifier given by the ADABOOST procedure in scikit-learn. We may replace ADABOOST by formulae (5) which yield the same result through simpler and less costly calculations: ADABOOST, from a computational point of view, is merely a formula.

The only thing needed is the prediction of the *weaklearners* and the resulting truth table from which the formula (5) follows readily. This remark is really important: the main role of ADABOOST is to choose the successive *weaklearners*. Once this choice of *weaklearners* is done, we describe a method to obtain the point of minimum of the convexified cost function, hence improving theoretically the resulting classifier.

References

[1] Peter L Bartlett, Michael I Jordan, and Jon D McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, 2006.

[2] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[3] Jean-Marc Brossier and Olivier Lafitte. Combining weak classifiers: a logical analysis. In *2021 23rd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 178–181. IEEE Computer Society, 2021.

[4] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[5] Christophe Giraud. *Introduction to high-dimensional statistics*. CRC Press, 2021.

[6] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

[7] Michael von Aitzing. *Thesaurus principum hac aetate in Europa viventium: quo progenitores eorum, tam paterni quam materni, simul ac fratres et sorores, inde ab origine reconduntur, usque ad annum a Christo nato...* Kempen, 1590.