 doi:10.20944/preprints202203.0362.v1

The Adaptive Reductions in Game Theory and their Applications to bOOsting

Ivano Azzini[1]

3 March 2022

Abstract. Starting from a very simple economic scenario, we build on it a game and then we introduce a general strategy able to reduce a regression problem to an equivalent binary classification problem. This reduction scheme (that we call *adaptive reduction* or also *dynamic reduction*) can be also used to derive a new boosting algorithm for regression problems named *bOOst*^{d}. The bOOst^{d} algorithm is very simple to implement, and it can use any learning algorithm with no priori assumptions. We present a *conjecture* for bOOst^{d} performances, which ensures a little error on training set. More important we can also provide a very good theoretical upper bound for the generalization error. We give a set of preliminary experimental results that seems to confirm our conjecture for bOOst^{d} performances on training set and the theoretical assumptions for the generalization error. We also provide a possible *justification* of why boosting often does *not* overfit. Finally, we leave some open problems and argue that in the future an adaptive single boosting (with an unique code) algorithm for binary, multi class and regression problems can be derived.

Keywords game theory, economic relation, problem reductions, binary and regression problems, machine learning, boosting, neural network.

1. Introduction

In this paper, starting from a simple economic scenario, we introduce a general strategy able to reduce a regression problem to an equivalent binary classification problem; we call this reduction scheme *adaptive reduction* or also *dynamic reduction*. Later, we observe as Freund & Schapire (1997) expose a first extension of AdaBoost for regression problems with an algorithm called AdaBoost.R, which attacks the regression problem by reducing it to a binary classification problem[2]. This approach has been analyzed and

[1] BriLeMa a no-profit association for Artificial Intelligence and Human Studies. Mail: ivano.azzini@yahoo.it.

[2] This is not the unique way to obtain boosting regression algorithms. There are many other authors who use different, effective, and very good techniques (greedy stage-wise minimization of a smooth cost function, barrier optimization, etc.) see for example Schapire, (2017); Meir & Rätsch, 2003; Duffy & Helmbold (2002). Still in this work we focus our attention only on this kind of solution.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

followed by many other authors (see for example Duffy & Helmbold, 2002; Meir & Rätsch, 2003; Bertoni, Campadelli & Parodi, 1997; Drucker, 1997; Ridgeway, Madigan, & Richardson 1999; Shrestha & Solomatine 2006). These researches show as this technique can be effective, but some questions are still open. For example AdaBoost.R is not simple to implement, and there is not a trivial way to generate a hypothesis whose loss is $\leq \frac{1}{2}$ as required by the algorithm. We observe as this problem is still unsolved, see for example Assaad, Bone' & Cardot (2006). Furthermore: "Unlike leveraging classifiers, leveraging regressors cannot always force the base regressor to output a useful function by simply modifying the distribution over the sample." Duffy & Helmbold (2002). More recent and similar analyses can be found also in Mendes-Moreira, Soares, Jorge, & Sousa, (2012), Beygelzimer, Hazan, Kale & Luo (2015) and Hanneke, Kontorovich, & Sadigurschi, M. (2018, 2019).

We use our new reduction scheme to obtain a new boosting algorithm for regression problems that attempts to overcome these limitations. We adopt the identical approach followed by Freund & Schapire, (1997) and others to derive the new boosting algorithm for regression problems. We call this algorithm *bOOst*^{d}. The bOOst^d algorithm is very simple to implement, and can use any "weak" or "base" learner without limitations. We give a conjecture for the bOOst^d performances that ensures a little error measure on training set. We also derive a very good theoretical upper bound for the generalization error. This is possible because we can use directly the method exposed in Freund & Schapire, (1997). We also argue that when boosting by *resampling* is used (Freund & Schapire, 1996), this generalization error limit can be better. In this way, we are able to provide also a possible justification of why boosting often does *not* overfit (Schapire, 2003).

Finally we give some preliminary experimental results, using neural networks as weak learner in learning functions context. The experiments with bOOst^d seem to confirm our theoretical framework.

2. What is the best that you can do?

Consider this simple economic and dynamic scenario: one of the major Oil Company in the world, to make its strategic decisions, requires predicting every day (or week, month,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

but also *every hour*) the price of the oil in many different world places. The Company assigns this job to the people in its statistical office (the Statistical Team). To win the annual award given by the company to the best statistician, the employees in the statistical office always do the best they can to meet the Company request. So we can ask: is it possible to find a good compromise (an equilibrium point) between the company demands and the statistician's ability to do prediction? In this context, what is the real skill of the statistician who will win the annual award?

This scenario admits a reverse, and more general interpretation within the game theory. Consider the following simple game. There are two players *Hillary* and *Albert*,

Hillary chooses an unknown random points y in [0, 1] and asks *Albert* to guess it.

Albert says: "It is impossible to do this! You must help me!"

Hillary … thinks a lot … the game can become very complicated for her …

Hillary: "Ok. I will provide you with the unknown random number y in [0, 1], after you can choose another real number d in [0, 1]. Then you must tell me if my y is minor of your d (in the same way $y>d$[3]) or not. We repeat this T times with different y and d at any time."

Albert: "Good, but I want to win more when my prediction is correct and d is close to zero. Is it all right for you?"

Hillary: "Ok. Given y and your d, if you correctly predict if $y<d$ you win 1/d otherwise you loose the same quantity".

In this game the goal (*Albert's* side) is to maximize the number of correct predictions for any t, trying to choose values of d just a little bit bigger than y, then the maximum that *Albert* can win is 1/y.

Certainly, Albert can always choose d = 1: he will always classify correctly $y<d$ but win 1 for any t. On the other side, given a unknown y, *Albert* can decide to have a bigger risk choosing very small d values… but if $y <d$ happens, he will make a great win which is equal to 1/d! *Albert* can also adopt a conservative strategy providing d values that are slightly deviations from ½, and so on. Note as if *Albert* always set d=1/2, we will have, an average loss equal to zero.

[3] In the following, for simplicity we consider only the situation in which we must predict if $y <d$ or not. The case in which one must predict if $y>d$ is identical.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

So *Hillary* is the above Oil Company. The value y is the unknown oil price in a specific market and *Albert* is a statistician. The "market's rules" force *Hilary* to have *Albert* as "adversary" and don't change it. Then d can be considered as a measure of *Albert*'s ability to predict the oil price but also an attempt to find a compromise (equilibrium) between the company request and the statistician's skill.

Hillary and *Albert* game can easily be generalized to predict n points at any t with a single d, or n points with n different d at any t … *Hillary* can know the value y[4], or in a more complex scenario *Hillary* can only sometime know the value y, but *Albert* doesn't know when this happens, and so on. With these generalizations, we have a set of statisticians (the Statistical Team) and we are able to represent the global market for the Oil Company. But, *we also have a good model for a learning process when a boosting algorithm is used.*

The booster is *Hillary* (or the Oil Company), the learner is *Albert* (or the people in the statistical office), y are the labels to predict[5] and d is, in general, a link between the booster's request and learner's ability (or *Hillary-Albert*, Oil Company-Statistical Office)[6]. Furthermore, we observe as the d value provided by *Albert* or the Statisticians *implicitly transforms the original regression problem in a binary classification problem*.

Can be useful to insert in the boosting process this paradigm? And if yes which is the best way to do it? To try to answer these questions and in order to create a model for the *Albert* and *Hillary* game (or Oil Company – Statistical Team relationship), we have introduced a dynamic reduction and then used it to derive a new boosting algorithms.

3. An adaptive scheme to reduce a continuous relation to a binary relation.

Mathematically we can proceed as follows: given two arbitrary sets X and Y a relation between X and Y is any set of couples (x, y). Formally, a relation is any subset of the Cartesian product: $X \times Y = \{(x, y) | x \in X, y \in Y\}$. Fixed the sets X and Y we can also create a collection of relations on the Cartesian product: $X \times Y = \{(x, y) | x \in X, y \in Y\}^t$, with t=1, 2,

[4] In Oil Company-Statistical Team relationship this happens when on a market is the Company itself that fixes the oil prize. The Statistical Team can know (or not) this information.

[5] Note as for the elements in the training set we know the y values.

[6] Or an attempt to extend the communication between booster and learner.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

3,… T. This is a very general scheme, but now we can fix: $X=\{x_1, x_2, ..., x_N\}$ and $Y=[0,1]$. In this case a relation is $X \times Y = \{(x_i, y_i) | x \in \{x_1, x_2, ..., x_N\}, y \in [0,1]\}$.

Now we reduce this set of couples in a binary relation with $Y=\{0,1\}$ in this way: fixed $d \in [0,1]$, and $\forall (x_i, y_i), y_i \in [0,1]$:

$$\tilde{y}_i = \begin{cases} 1 \; if \; y_i < d \\ 0 \; otherwise \end{cases}$$

Then our original set of couples are mapped in a set of binary couples $(x_i, \tilde{y}_i), \tilde{y}_i \in \{0,1\}$. In the same way if we have a collection of relations: fixed $d \in [0,1]$, $\forall (x_i, y_i)^t, y_i \in [0,1]$ and $\forall t$:

$$\tilde{y}_i = \begin{cases} 1 \; if \; y_i < d \\ 0 \; otherwise \end{cases}$$

We have $(x_i, \tilde{y}_i)^t, \tilde{y}_i \in \{0,1\}, \forall i$ and t=1,2,3, …T.

The value d splits the set $[0,1]$ into two parts, so it is possible to chose different d at any t, and change the reduction scheme as:

$\forall t$, *give me* a $d_t \in [0,1]$ and $\forall (x_i, y_i)^t, y_i \in [0,1]$, define

$$\tilde{y}_i = \begin{cases} 1 \; if \; y_i < d_t \\ 0 \; otherwise \end{cases}$$

so we obtain $(x_i, \tilde{y}_i)^t, \tilde{y}_i \in \{0,1\}$, $\forall i$ and t=1,2,3, …T.

We call this reduction scheme *adaptive* or also *dynamic* reduction: d*ynamic* because t can be viewed as the time; *adaptive* because for any t, the number d_t can be chosen randomly in [0, 1] but also found appropriately as a reaction to an external event or as an adjustment of the previous d_{t-1} value. In the following for simplicity we refer to the above scheme as *adaptive* reduction.

In the next paragraphs we describe as this methodology can be used to obtain a new boosting algorithm for regression problems.

4. Theoretical foundation of boosting approach to machine learning

Boosting is a widely accepted technique able to improve the performance of learning algorithms that are weakly accurate, and is considered to be an effective approach to

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

machine learning (Schapire, 2003; Schapire & Freund,2012). In this section we briefly review this approach, using the same learning framework described in (Freund & Schapire, 1997).

A learner receives N examples (x_i, y_i), y_i, i=1, …, N, usually called *Training Set* or *Sample Set*, chosen randomly according to some fixed, but unknown, distribution P on $X \times Y$, where each x_i belongs to some *domain X*, and each *label* y_i is in some label set *Y*. In *binary*, *multi-class* and *regression problems* or *settings*, we have respectively: $Y=\{0,1\}$, $Y=\{0,1,2,...,k\}$, $Y=[0,1]$. The goal is to learn to predict the label y given an instance x. Then the learner's job is to find a hypothesis h: $X \to Y$ to minimize the error: $\varepsilon = \Pr_{i \sim \mathcal{P}}[h(x_i) \neq y_i]$.

The goal of the *boosting* methodology is to produce a very accurate prediction rule using learning algorithms that are only moderately accurate ("weak" or also "base" learner, in the following WeakLearn).

Formally, boosting proceeds as follows: the boosting algorithm (or *booster*) is provided with a set of labeled training examples (x_i, y_i), and calls WeakLearn repeatedly. For each round t=1, …, T, the booster devises a distribution D_t over the set of examples, and asks WeakLearn for a weak hypothesis h_t with low error ε_t with respect to D_t (that is $\varepsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$). We call the quantity ε_t the *training set error* or *empirical error*. After T rounds, the booster must combine the weak hypotheses $h_1, ..., h_T$ into a single prediction rule (or *final hypothesis* h_f) that, hopefully, will be much more accurate than any weak hypothesis. The accuracy of the final hypothesis must also be related to the x_i that are in *X*, but not in the sample set. The final hypothesis error for these x_i, is called the *generalization error*, and is the most important quantity for the learning process.

After Freund and Schapire's work (1997), boosting has been the object of an intense research which has achieved much success. Many theoretical results and new boosting algorithms (for classification, multi-class and regression problems) have been discovered. A partial survey can be found, for example in Meir & Rätsch, (2003); Duffy & Helmbold (2002); Schapire (2017); Schapire & Freund (2012).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

5. The adaptive reductions and Adaboost.

Now we can use the above *adaptive* scheme to reduce a generic regression problem to a binary classification problem and then apply AdaBoost (Freund & Schapire, 1997).

We marked with tildes all the variables in the reduced space[7]. For each example (x_i, y_i), in the training set: $\{(x_i, y_i) \mid x_i \in X, y_i \in Y, i = 1,2,..., N\}$, $X = \{x_1, x_2, ..., x_N\}$ and $Y = [0,1]$, we defined a binary example as follows:

$$\tilde{x}_i \in \tilde{X} \text{ such that } \tilde{x}_i = x_i, \text{ with } x_i \in X \text{ . Then } X \equiv \tilde{X}.$$

$$\tilde{y}_i \in \tilde{Y} = \{0,1\}, \text{ such that } \tilde{y}_i = 1, \forall\, y_i \in Y.$$

Then, the training set is mapped in a set of binary couples all with label '1': ($\tilde{x}_i$, 1) ∀i.

Likewise, for the hypotheses generated by WeakLearn: given a value d∈[0,1], each hypothesis h: $X \to Y$, is reduced to a binary function $\tilde{h} : X \text{ x } Y \to \{0,1\}$ by the rule:

$$\tilde{h}(x, y) = \begin{cases} 1 \; if \; |h(x_i) - y_i| < \mathrm{d} \\ 0 \; otherwise \end{cases}$$

This is probably the easiest possible reduction for the hypotheses generated by WeakLearn. Each hypothesis $\tilde{h}$, simply *counts* the number of examples in the regression space for which: $|h(x) - y| < \mathrm{d}$.

Boosting methodology requires a distribution on the training set that is controlled by the booster as input (Freund & Schapire, 1997). During the boosting process, input distribution is modified by the booster at each round t = 1, …, T. This distribution is different from distribution P, which is on *X*x*Y* and unknown. Ordinarily, this distribution will be set to be uniform so that *D(i)*=1/N. For this distribution of probability on the examples our reduction is:

$$\tilde{D}(\tilde{i}) = D(\mathrm{i}) = 1/\mathrm{N}, \; \forall \mathrm{i} = 1, ..., \mathrm{N}.$$

Now we can calculate the binary error of $\tilde{h}$ with respect to density $\tilde{D}$. In the reduced space, AdaBoost computes:

$$\tilde{\varepsilon} = \sum_{i=1}^{N} \tilde{D}(i) \left| \tilde{h}(\tilde{x}_i) - \tilde{y}_i \right| = \sum_{i=1}^{N} \tilde{D}(i) \left| \tilde{h}(\tilde{x}_i) - 1 \right| = \sum_{i:\tilde{h}(x_i)=0} \tilde{D}(i)$$

[7] Also *Binary Classification Space* and *AdaBoost Space*.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

our reduction establishes that: $\left|\tilde{h}(\tilde{x}_i)-\tilde{y}_i\right| = 1 \Leftrightarrow \tilde{h}(\tilde{x})=0 \Leftrightarrow \left|h(x_i)-y_i\right| > \mathrm{d}$, then we have:

$$\tilde{\varepsilon} = \sum_{i:\left|h(x_i)-y_i\right|>\mathrm{d}} D(i) \qquad (1)$$

In the reduced space, the classification error is the number of weighted examples for which $\left|h(x)-y\right|>\mathrm{d}$.

At this point we can consider the *adaptive* reduction where the value of parameter d can be computed by the booster at every iteration. For example, for each t we can search the d_t number within the set: $\left\{\left|h_t(x_i)-y_i\right|, i=1,2,...,N\right\}$ by choosing the minimum d value for which the hypothesis error is < ½. Formally, $\forall$t we can found a set:

$$\hat{\mathrm{d}} = \min_i \left\{\left|h(x_i)-y_i\right| \textit{with } i=1,2,...,N, \textit{and such that } \tilde{\varepsilon}<1/2\right\} = \left|h(\hat{x}_i)-\hat{y}_i\right|.$$

By substituting in Eq. (1), and inserting the t dependence, we obtain:

$$\tilde{\varepsilon}_t = \sum_{i:\left|h_t(x_i)-y_i\right|>\hat{\mathrm{d}}_t} \tilde{D}_t(i) = \sum_{i:\left|h_t(x_i)-y_i\right|>\left|h_t(\hat{x}_i)-\hat{y}_i\right|} \tilde{D}_t(i) \le 1/2 \qquad (2)$$

We can see from Eq. (2) that d has disappeared and, since $(\hat{x}_i,\hat{y}_i)$ is in the training set, therefore the error only depends on the hypothesis generated by the learner at time t. Equation (2) removes the limitation introduced by AdaBoost.R and AdaBoost-RΔ (Freund & Schapire, 1997; Bertoni, Campadelli & Parodi, 1997). This is our first result: *we can use any learner without limitations,* or priori assumptions.

If we interpret this choice for d, within the Oil Company organization, this is a very simple "*equilibrium point"* determined by the people in the statistical office: "This is the best we can do!" For example: "The next hour the oil price will be < d (> d). We don't know how to do better!" Then for WeakLearn is: "With this training set this is the best I can do! Provide me with another training set, so maybe I can do better!" At this point, the Oil Company, and in a same way the booster are forced to choose this d value that enables them to continue in the learning process or to sale oil. We can also say that the collective skills for the people in the Statistical Team is equivalent to: we can only manipulate the distribution over the training set. In *Hillary-Albert* game that admits a reverse formulation the "*equilibrium point"* is fixed by *Albert:* "Now this is the most I can lose!"

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

We can also rewrite the equation (2) using the function HS(-), defined as HS(x)= 1 if x>0, zero otherwise.

$$\tilde{\varepsilon}_t = \sum_{i:|h_t(x_i)-y_i|>|h_t(\hat{x}_i)-\hat{y}_i|} \tilde{D}_t(i) = \sum_{i=1}^{N} \tilde{D}_i^t \cdot HS(|h_t(x_i)-y_i| - |h_t(\hat{x}_i)-\hat{y}_i|) = \sum_{i=1}^{N} \tilde{D}_i^t \cdot HS(|h_t(x_i)-y_i| - \mathrm{d}_t)$$

Finally in regression setting we focus our attention on the mean squared error (MSE). In particular, in the regression problems space, the empirical mean squared error is defined as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(h(x_i)-y_i)^2$$

MSE tends to zero (see Eq. (1)) when in the reduced space $\tilde{\varepsilon}$ and d tend to zero: $MSE \to 0 \Leftrightarrow (\tilde{\varepsilon} \to \wedge\, \mathrm{d} \to 0)$. So in order to reduce the MSE, the booster must do two tasks:

1. **to reduce the classification error**,
   and at the same time,
2. **to reduce d**.

We know as AdaBoost is very able to reduce the classification error ($\tilde{\varepsilon}$). At the same time the strategy used to choose d in Eq. (2) *guarantees* us that when we *simply* manipulate the distribution over the training set, it is not possible to do better (Duffy & Helmbold, 2002). In other words the method used to compute d in Eq. (2) ensures that to minimize the quantity $\tilde{\varepsilon}$ is equivalent to minimize the classification error and, at the same time, d. We cannot do more!

The reduction procedure is now completed, so we can apply AdaBoost in the reduced space to obtain a boosting algorithm for regression.

6. The bOOst^d Algorithm

The bOOst^d algorithm (Figure 1.) is built on the reduction exposed in Section 3. then: $\forall$ t, we use the *strength*, and then the *effects* obtained from the distribution modification on the training set (step 1 and step 3 in Figure 1.), in two ways:

first to improve the WeakLearn predictions as in AdaBoost and (simultaneously)

second to find the best possible reduction (reduction with d close to zero) in accordance with the hypothesis computed by WeakLearn.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

The bOOst^d algorithm has many similarities with AdaBoost: Input, Initialize, Steps 1 and 2 are identical. The greatest innovation of the bOOst^d algorithm is in step 3, where the booster takes control on the reduction choosing $d\ \forall t$. Parameter d is computed in accordance with our reduction (Eq. (2)), and its value is a function of the hypotheses returned by the learner.

In step 4 we compute a value β used to update the weights, then the weights are updated (Step 5.) in accordance with the method given in Step 4 and the reduction. Finally in figure 1, the final hypothesis is computed in accordance with our reduction.

Given the reduction methodology (Section 4.), we are also able to characterize theoretically the behavior of bOOst^d when used in a real learning task:

Conjecture 1

Suppose the weak learning algorithm WeakLearn, when called by bOOst^d, generates hypotheses with errors $\varepsilon_1,\ldots,\varepsilon_T$ (as defined in step 3 of figure 1).

Then the error $\varepsilon=\frac{1}{N}\sum_{i=1}^{N}HS(\left|h_f(x_i)-y_i\right|-\Lambda)$ of the final hypothesis h_f , output by bOOst^d is bounded above by: $\varepsilon\le\frac{1}{T-1}\left(\sum_{t=1}^{T}\varepsilon_t\right)$. Where Λ is the maximum in the sequence $\{d_1,...,d_T\}$.

We trust that this conjecture can be demonstrate using the work in Azzini (1998), where a full and complete demonstration of Theorem 12 in Freund & Schapire, (1997) is given. We show in section 7 that in general the error of bOOstd is very little and the limit for the error can be replaced with $\varepsilon\le\frac{1}{T-1}\left(\sum_{t=1}^{T}\varepsilon_t\right)^{\frac{nu}{de}}$, $de>nu$. We must use $de=nu=1$ only for f2 with the most simple neural network (a neural network with two neurons).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

Algorithm bOOst^{d}

Input: sequence of N labeled examples $\langle (x_1, y_1), \ldots, (x_N, y_N) \rangle$,

distribution D over the N examples,

weak learning algorithm WeakLearn,

integer T specifying number of iterations,

Initialize the weight vector $w_i^1 = D(i) = 1/N$, for i=1 . . .N

Do for t=1,2, . . .T**:**

1. Set

$$\mathbf{p}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^{N} w_i^t}$$

2. Call WeakLearn, providing it with the distribution $\mathbf{p}^t$; get back a hypothesis $h_t : X \to [0,1]$.

3. Calculate $d_t = \min_i \{|h_t(x_i) - y_i|\}$,

such that the error of $h_t : \varepsilon_t = \sum_{i=1}^{N} p_i^t HS(|h_t(x_i) - y_i| - d_t) < 1/2$.

4. Set $\beta_t = \frac{\varepsilon_t}{(1-\varepsilon_t)}$

5. Set the new weights vector to be: $w_i^{t+1} = w_i^t \cdot (\beta_t)^{1-HS(|h_t(x_i)-y_i|-d_t)}$

Output the hypothesis:

$$h_f(x) = \inf\left\{ y \in [y_i, 0] \cup [y_i, 1] : \sum_{t:h_t(x_i) \le y} (\log 1/(\beta_t)) \ge (1/2) \cdot \sum_t \log(1/(\beta_t)) \right\}$$

Figure 1: The bOOst^{d} Algorithm

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

Although if conjecture 1 (if true) guarantees that bOOst^d is able to reduce the error on the training set, we are interested at the generalization error. We have obtained bOOst^d by reducing a given regression problem to an equivalent binary classification problem. The final hypothesis generated by bOOst^d is obtained as a linear threshold of T binary functions, so all the results obtained for AdaBoost, as reported by Freund & Schapire, (1997) and Schapire, (2003), are also true for our work.

Furthermore in our experiments (Section 6), we use boosting by *resampling* (Freund & Schapire, 1996). When resampling method is used almost certainly WeakLearn will be trained (step 2 in Figure 1) only with a minimal part of the original input training set. We call the original input training set N, and N_t the training set used to *really* training WeakLearn and derived from N in accord with $\mathbf{P}_t$. Then we have $N_t \subseteq N$, but almost certainly $N_t \subset N$! Immediately after (step 3), the error is computed considering the entire training set. So the error, $\forall t = 1, ...,T$ can be considered composed of two quantities:

$$\varepsilon_t = \sum_{i \in N/N_t} p_i^t \mathrm{HS}(h_t(x_i) - d_t) + \sum_{i \in N_t} p_i^t \mathrm{HS}(h_t(x_i) - d_t) = \varepsilon_{TS_t} + \varepsilon_{G_t} \quad (4)$$

The quantity ε_{G_t} *in equation (4), can be considered as a kind of generalization error computed T times by bOOst^d*. If equation (4) is correct the generalization error for the final hypothesis can be better than the error predicted by the theory (Freund & Schapire, 1997 and Schapire, 2003), furthermore equation (4) can also explain why boosting often does *not* overfit (Schapire, 2003).

Finally we observe as bOOst^d is entirely built on the *Hillary-Albert* game (or *Oil Company-Statistical Office* relations): it is an attempt to find an "*equilibrium point"*, as described in section 4. Most probably the "*equilibrium point"* found by bOOst^d is naïve, nevertheless when the booster *simply* manipulates the distribution over the training set to force the learning process, it is not possible to do better. In despite of that the experimental results are very good.

All the points listed in Figure 1. can be interpreted within the *Hillary-Albert* game or *Oil Company-Statistical Office* relations. For example in the *Oil Company-Statistical Office* case, the modality used by the booster to change the distribution on the training set (step

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

5. in Figure 1, Freund & Schapire, 1997) can be interpreted as: statisticians who have worked well at time t-1, now are working less (they are resting), and vice versa: statisticians who have worked worse, now work more.

7. Experiments with bOOst^d.

In order to evaluate and to better understand bOOst^d and in general our methodology, we have performed some experiments. To have a sound testing framework we repeat the same experiments performed by many predecessors.

We have easily implemented bOOst^d in MATLAB/Octave environment (Matlab 2013, Octave 2018), following step by steps the Figure 1. We can use any learner without any *a prior* assumption; nevertheless here we use as WeakLearner the very basic backpropagation n-layer feed-forward neural network.

For example in the following pages, a neural network N: 1-2-2-1 is a backpropagation 2-layer feed-forward neural network with 2 neurons in the first and second layer. Transfer functions for the entire hidden layers are Log-Sigmoid and the linear function for the output layer and the Levenberg Marquardt backpropagation algorithm is used. *In other word, if there is no other specifications, we use the Matlab Neural Network toolbox/Octave neural network package without changes from the default setting.*

As we can use any unconstrained learner, a stopping criterion for the parameter T is required, or simply T can be fixed a priori. We choose for simplicity the second solution. Then we test bOOst^d with very simple neural network, but also with more complex nets and with different T values.

These WeakLearner are used in learning functions task (functions approximation or also interpolation) for a set of function: $\{f_i\}: R \to R$.

In all experiments we use a training set with N=200 examples and a test set of N=100 examples. We use the same training and test set to compare the performances of backpropagation without boosting and bOOst^d. These points are extracted randomly (in accord with the uniform distribution) from the set X and then we have shifted the relative function values in the set Y=[0,1] as required by bOOst^d.

For any test we always repeat the experiment 100 times and provide the necessary statistics.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

The functions set $\{f_i\}$ considered here is:

- $f_1(x)=\frac{\sin(10\cdot x)+2}{4}+\frac{\sin\left(50\cdot(x+0.5)^2\right)}{15}+0.1\cdot N(0,0.1)$, $0 \le x \le 1$ from Bertoni, Campadelli & Parodi, 1997.
- $f_2(x)=+\frac{\sin\left(\frac{1}{0.03\cdot x}\right)}{5}+0.2\cdot N(0,0.2)$, $0 \le x \le 1$ from Bertoni, Campadelli & Parodi, 1997.
- $f_3(x)=(x+1)\cdot\exp(-3\cdot x+3)$, $0 \le x \le 1$ from Zainuddin & Pauline, 2008.
- $f_4(x)=\sin(4\cdot\pi\cdot x)\cdot\exp(-|5\cdot x|)$, $-1 \le x \le 1$ from Zainuddin & Pauline, 2008.
- $f_5(x)=\begin{cases}-2.186\cdot x-12.864 & \text{if } -10 \le x < -2\\ -4.264\cdot x & \text{if } -2 \le x < -0\\ 10\cdot e^{-0.05\cdot x-0.5}\cdot\sin\left[(0.03\cdot x+0.7)\cdot x\right] & \text{if } 0 \le x < 10\end{cases}$

  is a piecewise continuous function from Zainuddin & Pauline, 2008.
- $f_6(x)=\frac{e^{c1\cdot x^{c2}}}{2\cdot\pi^{c3}\cdot x^{c4}}$ $-5 \le x < 5$, with c1=-2/3, c2=3/2, c3=1/2 and c4=1/4.

The latest function f_6 is a variant of the asymptotic formulae of the Airy functions, Abramowitz & Stegun (1970) with x shifted in the set [-5, 5], and only the real part of y is considered. This function has an interesting plot, if considered in learning environment, we call this function MaGa function. And

- $f_7(x)=4\cdot\exp(\sin(x))+N(0,1)$ $-0 \le x < 20$
- $f_8(x)=x\cdot\cos^2\left(\frac{x}{2}\right)+N(0,1)$ $-0 \le x < 20$

from Rutkowski, Jaworski, & Duda, 2019.

7.1 Experimental Results

7.1.1. The bOOst^d error on training set.

In our first experimental session, we verify the ability of bOOst^d to reduce error on the training set, in binary space and then (we hope) the relative MSE. To do this we consider

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

the function f_2. In the interval [0,1], f_2 can be considered as the most "*hard*" function in $\{f_i\}$ (see figure 1). On the other functions in $\{f_i\}$ our method outperforms the performances for f_2 reported here.

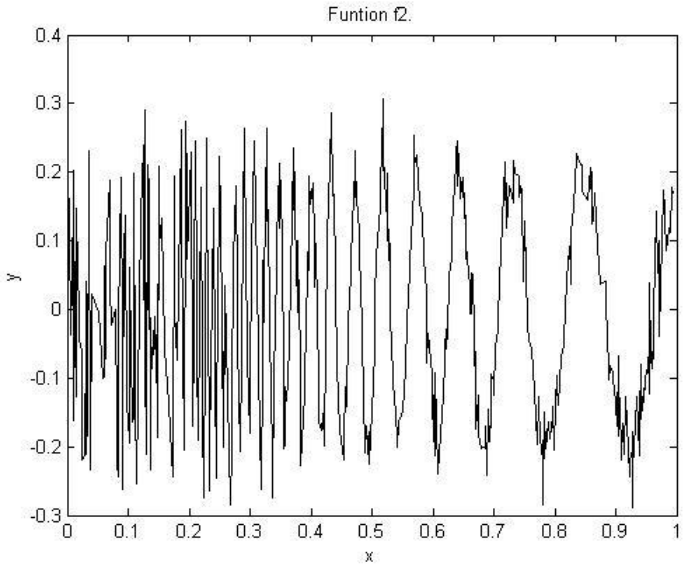

Figure 1. Function f2.

Using a 1-10-10-1 neural network (Bertoni, Campadelli & Parodi, 1997) on this function we obtain the results in Table 1.

NNa **1-10-10-1** Ripetitions=100	meanε	mean MSE
T=2	0.2794	0.015851
T=5	0.2228	0.01293 (**0.0428** in *Bertoni et Al*)
T=10	0.1987	0.012429

Table 1.

In Table 1, we can see as bOOst^d is able to reduce the errors on training set. Furthermore, in (Bertoni, Campadelli & Parodi, 1997) is reported the mean MSE on training set for T=5, using the same network. Our method reduces the error on training set by a factor

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

$\frac{0.0428}{0.01293} = 3.3101$ if compared with ADABOOST-RΔ. Than we can consider bOOst^d as an effective methodology.

7.1.2. Is Conjecture 1 supported experimentally?

We want also check if the Conjecture 1 is valid. To do this we use in our experiments the most simple neural network: a network with only two neurons in one hidden layer. In general, bOOst^d performs much better than it is predicted in Conjecture 1. The limit in conjecture is basically due to the presence of f_2 in $\{f_i\}$, which is a very "*hard*" function. In all other cases we can rewrite Conjecture 1 with a more appropriate limit for the bOOst^d error.

The results relative to f_2 are showed in Table 1. For the 100 different training sets considered, the Conjecture 1 was always true. The error of final hypothesis as computed in Conjecture 1, decreases when T increases. The MSE is more or less the same when T increases.

NNa **1-2-1** Ripetitions=100	$\max \varepsilon$	$\max \frac{1}{T-1}\sum_{t=1}^{T} \varepsilon_t$	mean MSE
T=2	0.50	0.99455	0.022034
T=5	0.515	0.62339	0.020807
T=10	0.495	0.55418	0.020743
T=15	0.485	0.53377	0.020881
T=20	0.475	0.52474	0.021007
T=30	0.455	0.51531	0.020797
T=50	0.43	0.5083	0.020938

Table 2. Function f_2

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

Also for the other functions in $\{f_i\}$, Conjecture 1 is always true, the difference in these cases is that bOOst^d has the ability to drastically reduce ε and MSE. In Table 3 and 4, we report[8] the results for other two functions in $\{f_i\}$.

NNa **1-2-1** Ripetitions=100	$\max \varepsilon$	$\max \frac{1}{T-1}\sum_{t=1}^{T} \varepsilon_t$	mean MSE
T=2	0.49	0.9949	8.2469
T=5	0.345	0.62201	6.7202
T=10	0.285	0.55352	6.2778
T=15	0.265	0.53379	5.5457
T=20	0.258	0.52421	5.5296
T=30	0.23	0.5151	5.5184
T=50	0.22	0.50809	5.4382

Table 3. Function f_5.

NNa **1-2-1** Ripetitions=100	$\max \varepsilon$	$\max \frac{1}{T-1}\sum_{t=1}^{T} \varepsilon_t$	mean MSE
T=2	0.5	0.99285	11.0681
T=5	0.25	0.6223	9.3011
T=10	0.225	0.55368	9.2544
T=15	0.205	0.53382	9.0901
T=20	0.19	0.52435	9.0949
T=30	0.175	0.51533	9.0763
T=50	0.15	0.50809	9.0153

Table 4. Function f_8.

[8] For simplicity we do not report here the tables for functions 1,3,4,6,7 because we have similar and better results.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

7.1.3. Generalization Error

Seem that bOOst^d has good performances on training set, but we are mainly interested to the generalization error of bOOst^d. So now, given an *a priori* T (we choose T=5), a fixed neural network architecture, we use the same training set to train a single backpropagation neural network and to build the neural network committee required in bOOst^d. Finally we compare the generalization error (using the same test set) for these two strategies, for the function set $\{f_i\}$. Results are in Table 5-7, for three different neural network architectures. We can see as bOOst^d is effective in the reduction of generalization error also for the most simple neural network.

NNa 1-2-1 Repetitions=100	Error on Test Set **(Back propagation Without Boosting)**	Error on Test Set **(bOOst^d with T=5)**	$\frac{\varepsilon_{NN}}{\varepsilon_{bOOst^d}}$
f1	0.012053	0.01046	1.1523
f2	0.021854	0.021655	1.0092
f3	42.232	22.3785	1.8872
f4	0.029257	0.025434	1.1503
f5	7.9564	6.7898	1.1718
f6	0.012478	0.0079035	1.5788
f7	9.3053	9.092	1.0235
f8	10.7265	9.6053	1.1167

Table 5.

NNa 1-2-2-1 Ripetitions=100	Error on Test Set **(Back propagation Without Boosting)**	Error on Test Set **(bOOst^d with T=5)**	$\frac{\varepsilon_{NN}}{\varepsilon_{bOOst^d}}$
f1	0.010153	0.0069136	1.4685
f2	0.021739	0.021514	1.0105
f3	41.7576	13.042	3.2018
f4	0.029512	0.024939	1.1834
f5	9.9774	8.8067	1.1329

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

f6	0.0096695	0.0065145	1.4843
f7	8.9486	8.6654	1.0327
f8	11.4917	10.407	1.1042

Table 6.

NNa 1-5-5-1 Repetitions=100	Error on Test Set **(Back propagation Without Boosting)**	Error on Test Set **(bOOst^d with T=5)**	$\frac{\varepsilon_{NN}}{\varepsilon_{bOOst^d}}$
f1	0.00248	0.0023925	1.0366
f2	0.020977	0.020483	1.0241
f3	1.9450	0.00023952	~ 8120
f4	0.0043313	1.1046e-05	~ 392
f5	1.0885	0.014141	~ 77
f6	0.0016568	0.0011594	1.429
f7	0.91164	0.017739	~51
f8	1.4678	0.10886	~13.5

Table 7.

We can see (Table 7) as, in this case, the reduction in generalization error is impressive.

8 Conclusion and open problems.

In this work we have introduced a new method to reduce regression problems to binary classification problems. Using this strategy we have derived a new boosting algorithm for regression problems: bOOst^d. The new algorithm exhibits good performances using neural network in learning function, nevertheless some questions remain still open.

To validate the bOOst^d features, other experiments must be done using different weak leaners and in other learning contexts.

Conjecture 1 must be demonstrated and eventually improved, in accordance with experimental results.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 March 2022 doi:10.20944/preprints202203.0362.v1

From a theoretical point of view, bOOst^d can generate infinite hypotheses, then can be very interesting to find a theoretical relation between T and the generalization error.

The rule used to compute beta ($\beta_t = \frac{\varepsilon_t}{(1-\varepsilon_t)}$), is the only possible here? Or other rules that take also in account d_t can be used? i.e. $\beta_t = f\left(d_t, \frac{\varepsilon_t}{(1-\varepsilon_t)}\right)$ or in general $\beta_t = f(d_t, \varepsilon_t)$. What is the best strategy to play the *Hillary-Albert* game?

Perhaps, the most interesting aspect of bOOst^d is that its pseudo-code is essentially equal to the Adabost code; therefore, in principle, we can have an unique boosting algorithm for both regression and classification problems.

From this observation we can argue that a bOOst^d similar algorithm for multiclass classification problems can be derived. If this algorithm exists, we can have an unique adaptive boosting algorithm for binary multi class and regression problems. An algorithm like this can exhibit an interesting behavior (similar to human abilities): after a preliminary analysis (or also an on-line analysis of examples in training set), it can adapt itself to manage the appropriate learning problem. We assume that a *single boosting algorithm able to adapt itself to the specific learning environment can be very usefully.*

For example, when d is chosen in the open set $(0,1)$ bOOst^d is able to predict label $y \in [0,1]$. If we set d = 1 (or d=0) at the beginning of the boosting process, bOOst^d is reduced to a boosting algorithm for binary classification problems (i.e. we have $y \in \{0,1\}$), and it becomes *identical* to AdaBoost. Is it possible to "reduce" bOOst^d to manage multi-class classification problems[9]?

A trivial solution can be built on the following observation: for many multi-class classification problems (Y = { 1, 2, …, k-1, k }), when the learner misclassifies an example, the error can have different levels of *severity*. For example in a multi-classification task for vehicles, if the learner classifies a car as a truck, it is much more serious than a misclassification between two different types of car. Furthermore if a

[9] In the same way it is possible to change the *Hillary–Albert* game and the *Oil Company-Statistichal Office* interaction when $y \in Y = \{1,2,...,k\}$?

 doi:10.20944/preprints202203.0362.v1

learner misclassifies two station wagons it is different if it misclassifies a coupé car and a station wagon.

For this reason, for some learning tasks can be possible to define an *order* on the set Y. In these cases, we can put the elements of Y in correspondence to the interval [0,1] so that close labels are "*similar*" and distant labels are "*different*".

After this association we have a new label set Y' = {1', 2', …, (k-1)', k'}, obtained from a permutation of the Y elements, which are now ordered: k'>(k-1)'> … >2'>1', in accordance with a "similarity" measure which is intrinsic to the learning problem considered. When this is possible we can use d like in regression setting.

Acknowledgment

The author would like to thank Manuela Fabbri, Rossana Rosati and Emanuele (Lele) Sassi for their encouragement and comments that greatly contributed to the manuscript.

Bibliography

Abramowitz, Milton; Stegun, Irene A. (1970), *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, New York: Dover Publications, Ninth printing.

Assaad, M., Bone', R., Cardot, H. (2006). A new boosting algorithm for improved time-series forecasting with recurrent neural networks. *Information Fusion*, Volume 9, Issue 1, Pages 41-55 ISSN:1566-2535.

Azzini, I. (1998). bOOsting: a methodology to improve the performances of learning algorithms. Master's Degree Thesis.

Bertoni, A., Campadelli, P., & Parodi, M. (1997). A boosting algorithm for regression. In W.Gerstner, A.Germond, M.Hasler, and J. D. Nicoud (Eds.), *Proceedings ICANN'97, Int. Conf. on Artificial Neural Networks*, (pp 343-348). Berlin, Germany: Springer-Verlag. Vol. V of LNCS.

Beygelzimer, A., Hazan, A., Kale, S., and Luo, H. (2015). Online gradient boosting. In *Advances in Neural Information Processing Systems*, pp. 2458–2466.

Drucker, H. (1997, July). Improving regressors using boosting techniques. In *ICML* (Vol. 97, pp. 107-115).

Duffy N., & Helmbold, D. (2002) Boosting Methods for Regression, *Machine Learning* 47:2-3, 153-200.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. in *Icml* 96, 148–156 (Bari, Italy).

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:1, 119-139.

Hanneke, S., Kontorovich, A., & Sadigurschi, M. (2018). Sample Compression for Real-Valued Learners. *arXiv preprint arXiv:1805.08254*.

Hanneke, S., Kontorovich, A., & Sadigurschi, M. (2019). Sample Compression for Real-Valued Learners. Proceedings of Machine Learning Research vol 98:1–23, 2019 30th International Conference on Algorithmic Learning Theory.

Matlab (2013). A MathWorks software http://www.mathworks.com.

 doi:10.20944/preprints202203.0362.v1

Meir, R. & Rätsch, G. (2003). An introduction to boosting and leveraging. In S. Mendelson and A. Smola, (Eds.), *Advanced Lectures on Machine Learning*, LNCS (pp 119-184). New York, NY: Springer-Verlag.

Mendes-Moreira, J., Soares, C., Jorge, A. M., & Sousa, J. F. D. (2012). Ensemble approaches for regression: A survey. *ACM Computing Surveys (CSUR)*, *45*(1), 10.

Octave (2018). The GNU Octave https://www.gnu.org/software/octave.

Rätsch, G., Demiriz, A. & Bennett, K. P. (2002). Sparse regression ensembles in infinite and finite hypothesis spaces. *Machine Learning*, 48:1-3, 189-218.

Ridgeway, G., Madigan, D., & Richardson, T. (1999, January). Boosting methodology for regression problems. In *AISTATS*.

Rutkowski, L., Jaworski, M., & Duda, P. (2019). Stream Data Mining: Algorithms and Their Probabilistic Properties (Vol. 56). Heidelberg: Springer.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5:2, 197-227.

Schapire, R. E. (2003). The boosting approach to machine learning: An overview. In D.D. Dennison, M. H. Hansen, C. Holmes, B. Mallick & B. Yu (ed.) *MSRI Workshop on Nonlinear Estimation and Classification* (pp. 149-172). New York, NY: Springer-Verlag.

Schapire R.E., Freund Y. (2012). Boosting: Foundations and algorithms. MIT press.

Schapire, R. E. (2017). http://rob.schapire.net/publist.html.

Shrestha D. L., D. P. Solomatine (2006). Experiments with AdaBoost.RT, an Improved Boosting Scheme for Regression. *Neural Computation,* 18:1678-1710. The MIT Press.

Zainuddin, Z. Pauline, O. (2008). Function approximation using artificial neural networks *WSEAS Transactions on Mathematics*, Volume 7 , Issue 6 (June 2008) Pages 333-338 ISSN:1109-2769.