Multi-class boosting with asymmetric binary weak-learners

Antonio Fernández-Baldera, Luis Baumela

Departamento de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus Montegancedo s/n
28660 Boadilla del Monte, Spain
Corresponding author email: `lbaumela@fi.upm.es`
Corresponding autor phone: +34913367440
Corresponding autor fax: +34913524819

Abstract

We introduce a multi-class generalization of AdaBoost with binary weak-learners. We use a vectorial codification to represent class labels and a multi-class exponential loss function to evaluate classifier responses. This representation produces a set of margin values that provide a range of punishments for failures and rewards for successes. Moreover, the stage-wise optimization of this model introduces an asymmetric boosting procedure whose costs depend on the number of classes separated by each weak-learner. In this way the boosting procedure takes into account class imbalances when building the ensemble. The experiments performed compare this new approach favorably to AdaBoost.MH, GentleBoost and the SAMME algorithms.

Keywords:
AdaBoost, multi-class classification, asymmetric binary weak-learners, class imbalance

1. Introduction

Boosting algorithms are learning schemes that produce an accurate or *strong classifier* by combining a set of simple base prediction rules or *weak-learners*. Their popularity is based not only on the fact that it is often much easier to devise a simple but inaccurate prediction rule than building a highly accurate classifier, but also because of the successful practical results and good theoretical properties of the algorithm. They have been extensively

Preprint submitted to Pattern Recognition *November 11, 2013*

used for detecting [1, 2, 3, 4] and recognizing [5, 6] faces, people, objects and actions [7] in images. The boosting approach works in an iterative way. First a weight distribution is defined over the training set. Then, at each iteration, the best weak-learner according to the weight distribution is selected and combined with the previously selected weak-learners to form the strong classifier. Weights are updated to decrease the importance of correctly classified samples, so the algorithm tends to concentrate on the "difficult" examples.

The most well-known boosting algorithm, AdaBoost, was introduced in the context of two-class (binary) classification, but it was soon extended to the multi-class case [8]. Broadly speaking, there are two approaches for extending binary Boosting algorithms to the multi-class case, depending on whether multi-class or binary weak-learners are used. The most straightforward extension substitutes AdaBoost's binary weak-learners by multi-class ones, this is the case of AdaBoost.M1, AdaBoost.M2 [8], J-classes LogitBoost [9], multi-class GentleBoost [10] and SAMME [11]. The second approach transforms the original multi-class problem into a set of binary problems solved using binary weak-learners, each of which separates the set of classes in two groups. Shapire and Singer's AdaBoost.MH algorithm [12] is perhaps the most popular approach of this kind. It creates a set of binary problems for each sample and each possible label, providing a predictor for each class. An alternative approach is to reduce the multi-class problem to multiple binary ones using a codeword to represent each class label [13, 14, 15]. When training the weak-learners this binarization process may produce imbalanced data distributions, that are known to affect negatively in the classifier performance [16, 17]. None of the binary multi-class boosting algorithms reported in the literature address this issue.

Another aspect of interest in multi-class algorithms is the codification of class labels. Appropriate vectorial encodings usually reduce the complexity of the problem. The encoding introduced in [18] for building a multi-class Support Vector Machine (SVM), was also used in the SAMME [11] and GAMBLE [19] algorithms and is related to other margin-based methods [10]. Shapire uses Error Correcting Output Codes for solving a multi-class problem using multiple binary classifiers [13, 12]. Our proposal uses vectorial encodings for representing class labels and classifier responses.

In this paper we introduce a multi-class generalization of AdaBoost that uses ideas present in previous works. We use binary weak-learners to separate groups of classes, like [15, 13, 12], and a margin-based exponential loss function with a vectorial encoding like [18, 11, 19]. However, the final result is new. To model the uncertainty in the classification provided by each weak-learner we use different vectorial encodings for representing class labels and classifier responses. This codification yields an asymmetry in the evaluation of classifier performance that produces different margin values depending on the number of classes separated by each weak-learner. Thus, at each boosting iteration, the sample weight distribution is updated as usually according to the performance of the weak-learner, but also, depending on the number of classes in each group. In this way our boosting approach takes into account both, the uncertainty in the classification of a sample in a group of classes, and the imbalances in the number of classes separated by the weak-learner [16, 17]. The resulting algorithm is called *PIBoost*, which stands for Partially Informative Boosting, reflecting the idea that the boosting process collects partial information about classification provided by each weak-learner.

In the experiments conducted we compare two versions of PIBoost with GentleBoost [9], AdaBoost.MH [12] and SAMME [11] using 15 databases from the UCI repository. These experiments prove that one of PIBoost versions provides a statistically significant improvement in performance when compared with the other algorithms.

The rest of the paper is organized as follows. Section 2 presents the concepts from binary and multi-class boosting that are most related to our proposal. In Section 3 we introduce our multi-class margin expansion, based on which in section 4 we present the PIBoost algorithm. Experiments with benchmark data are discussed in Section 5. In Section 6 we relate our proposal with others in the literature and in Section 7 we draw conclusions. Finally, we give the proofs of some results in two Appendices.

2. Boosting

In this section we briefly review some background concepts that are directly related to our proposal. Suppose we have a set of N labeled instances $\{(\mathbf{x}_i, l_i)\}, i = 1, \ldots, N$; where $\mathbf{x}_i$ belongs to a domain X and l_i belongs to $L = \{1, 2, \ldots, K\}$, the finite label set of the problem (when $K = 2$ we simply use $L = \{+1, -1\}$). Henceforth the words *label* and *class* will have the same meaning. $\mathcal{P}(L)$ will denote the power-set of labels, i.e. the set of all possible subsets of L. We will use capital letters, e.g. $T(\mathbf{x})$ or $H(\mathbf{x})$, for denoting weak or strong classifiers that take values on a finite set of values, like L. Small bold letters, e.g. $\mathbf{g}(\mathbf{x})$ or $\mathbf{f}(\mathbf{x})$, will denote classifiers that take value on a set of vectors.

2.1. Binary Boosting

The first successful boosting procedure was introduced by Freund and Schapire with their AdaBoost algorithm [8] for the problem of binary classification. It provides a way of combining the performance of many weak classifiers, $G(\mathbf{x}) : X \rightarrow L$, here $L = \{+1, -1\}$, to produce a powerful "committee" or strong classifier

$$H(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m G_m(\mathbf{x}),$$

whose prediction is $\text{sign}(H(\mathbf{x}))$.

AdaBoost can also be seen as a stage-wise algorithm fitting an additive model [9, 20]. This interpretation provides, at each round m, a *direction* for classification, $G_m(\mathbf{x}) = \pm 1$, and a *step size*, α_m, the former understood as a sign on a line and the latter as a measure of confidence in the predictions of G_m.

Weak-learners G_m and constants α_m are estimated in such a way that they minimize a *loss function* [9, 12]

$$\mathcal{L}\left(l, H(\mathbf{x})\right) = \exp(-lH(\mathbf{x}))$$

defined on the value of $z = lH(\mathbf{x})$ known as *margin* [15, 10].

To achieve this a weight distribution is defined over the whole training set, assigning each training sample $\mathbf{x}_i$ a weight w_i. At each iteration, m, the selected weak-learner is the best classifier according to the weight distribution. This classifier is added to the ensemble multiplied by the goodness parameter α_m. Training data $\mathbf{x}_i$ are re-weighted with $\mathcal{L}(l, \alpha_m G_m(\mathbf{x}))$. So, the weights of samples miss-classified by G_m are multiplied by e^{α_m}, and are thus increased. The weights of correctly classified samples are multiplied by $e^{-\alpha_m}$ and so decreased (see Algorithm 1). In this way, new weak-learners will concentrate on samples located on the frontier between the classes. Other loss functions such as the Logit [9], Squared Hinge [10] or Tangent loss [21] have also been used for deriving alternative boosting algorithms.

Note here that there are only two possible margin values ± 1 and, hence, two possible weight updates $e^{\pm \alpha_m}$ in each iteration. In the next sections, and for multi-class classification problems, we will introduce a vectorial encoding that provides a margin interpretation that has several possible values, and thus, various weight updates.

Algorithm 1 : AdaBoost

```
Initialize the weight Vector W with uniform distribution ω_i = 1/N , i = 1, ..., N.
for m = 1 to M do
    Fit a classifier G_m(x) to the training data using weights W.
    Compute weighted error: Err_m = Σ_{i=1}^{N} ω_i I (G_m(x_i) ≠ l_i).
    Compute α_m = (1/2) log ((1 − Err_m)/Err_m).
    Update weights ω_i ← ω_i · exp (−α_m l_i G_m(x_i)) , i = 1, ..., N.
    Re-normalize W.
end for
Output Final Classifier: sign ( Σ_{m=1}^{M} α_m G_m(x) )
```

2.2. Multi-class boosting with vectorial encoding

A successful way to generalize the symmetry of class-label representation in the binary case to the multi-class case is using a set of vector-valued class codes that represent the correspondence between the label set $L = \{1, \dots, K\}$ and a collection of vectors $Y = \{\mathbf{y}_1, \dots, \mathbf{y}_K\}$, where vector $\mathbf{y}_l$ has a value 1 in the l-th co-ordinate and $\frac{-1}{K-1}$ elsewhere. So, if $l_i = 1$, the code vector representing class 1 is $\mathbf{y}_1 = \left(1, \frac{-1}{K-1}, \dots, \frac{-1}{K-1}\right)^\top$. It is immediate to see the equivalence between classifiers $H(\mathbf{x})$ defined over L and classifiers $\mathbf{f}(\mathbf{x})$ defined over Y:

$$H(\mathbf{x}) = l \in L \;\Leftrightarrow\; \mathbf{f}(\mathbf{x}) = \mathbf{y}_l \in Y. \tag{1}$$

This codification was first introduced by Lee, Lin and Wahba [18] for extending the binary SVM to the multi-class case. More recently Zou, Zhu and Hastie [10] generalize the concept of binary margin to the multi-class case using a related vectorial codification in which a K-vector $\mathbf{y}$ is said to be a *margin vector* if it satisfies the *sum-to-zero* condition, $\mathbf{y}^\top \mathbf{1} = 0$, where $\mathbf{1}$ denotes a vector of ones. This sum-to-zero condition reflects the implicit nature of the response in classification problems in which each y_i takes one and only one value from a set of labels.

The SAMME algorithm generalizes the binary AdaBoost to the multi-class case [11]. It also uses Lee, Lin and Wahba's vector codification and a multi-class exponential loss that is minimized using a stage-wise additive gradient descent approach. The exponential loss is the same as the original binary exponential loss function and the binary margin, $z = lG(\mathbf{x})$, is replaced by the multi-class vectorial margin, defined with a scalar product

$z = \mathbf{y}^\top \mathbf{f}(\mathbf{x})$; i.e.

$$\mathcal{L}\left(\mathbf{y}, \mathbf{f}(\mathbf{x})\right) = \exp\left(-\frac{\mathbf{y}^\top \mathbf{f}(\mathbf{x})}{K}\right). \quad (2)$$

Further, it can be proved that the population minimizer of this exponential loss, $\arg\min_{\mathbf{f}(\mathbf{x})} E_{\mathbf{y}|X=\mathbf{x}}\left[\mathcal{L}\left(\mathbf{y}, \mathbf{f}(\mathbf{x})\right)\right]$, corresponds to the multi-class Bayes optimal classification rule [11]

$$\arg\max_k f_k(\mathbf{x}) = \arg\max_k Prob\left(Y = y_k|\mathbf{x}\right).$$

Other loss functions, such as the logit or L_2, share this property and may also be used for building boosting algorithms.

In the proposal that we introduce in the next section we generalize the class-label representation here described so that our boosting algorithm can model the asymmetries arising in the binary classifications performed by the weak-learners. Although other loss functions could have been used, we will use the exponential loss to maintain the similarity with the original AdaBoost algorithm.

3. Multi-class margin expansion

The use of margin vectors for coding data labels and the labels estimated by a classifier introduces a natural generalization of binary classification, in such a way that new margin-based algorithms can be derived. In this section we introduce a new multi-class margin expansion. Similarly to [18, 10, 11] we use sum-to-zero margin vectors to represent multi-class membership. However, in our proposal, data labels and those estimated by a classifier will not be defined on the same set of vectors. This will produce, for each iteration of the algorithm, different margin values for each sample, depending on the number of classes separated by the weak-learner. This is related to the asymmetry produced in the classification when the number of classes separated by a weak-learner is different on each side and to the "difficulty" or information content of that classification.

The essence of the margin approach resides in maintaining negative/positive values of the margin when a classifier has respectively a failure/success. That is, if $\mathbf{y}, \mathbf{f}(\mathbf{x}) \in Y$ the margin $z = \mathbf{y}^\top \mathbf{f}(\mathbf{x})$ satisfies: $z > 0 \Leftrightarrow \mathbf{y} = \mathbf{f}(\mathbf{x})$ and $z < 0 \Leftrightarrow \mathbf{y} \neq \mathbf{f}(\mathbf{x})$. We extend the set Y by allowing that each $\mathbf{y}_l$ may also take a negative value, that can be interpreted as a fair vote for *any label but the l-th*. This vector encodes the uncertainty in the classifier response, by evenly dividing the evidence among all classes, but the l-th. It provides the smallest amount of information about the classification of an instance;

i.e. a negative classification, the instance does not belong to class l but to any other. Our goal is to build a boosting algorithm that combines both positive and negative weak responses into a *strong decision*.

We introduce new kinds of margin vectors through fixing a group of s-labels, $S \in \mathcal{P}(L)$, and defining $\mathbf{y}^S$ in the following way:

$$\mathbf{y}^S = \left(y_1^S, \ldots, y_K^S\right) \text{ with } y_i^S = \begin{cases} \frac{1}{s} & \text{if } i \in S \\ \frac{-1}{K-s} & \text{if } i \notin S \end{cases} \tag{3}$$

It is immediate to see that any $\mathbf{y}^S$ is a margin vector in the *sum-to-zero* sense [18, 10]. In addition, if S^c is the complementary set of $S \in \mathcal{P}(L)$, then $\mathbf{y}^{S^c} = -\mathbf{y}^S$. Let $\hat{Y}$ be the whole set of vectors obtained in this fashion. We want to use $\hat{Y}$ as arrival set, that is: $\mathbf{f} : X \to \hat{Y}$, but under a binary perspective. The difference with respect to other approaches using similar codification [18, 10, 11] is that the correspondence defined in (1) is broken. In particular, weak-learners will take values in $\left\{\mathbf{y}^S, -\mathbf{y}^S\right\}$ rather than the whole set $\hat{Y}$. The combination of answers obtained by the boosting algorithm will provide complete information over $\hat{Y}$. So now the correspondence for each weak-learner is binary

$$H^S(\mathbf{x}) = \pm 1 \;\Leftrightarrow\; \mathbf{f}^S(\mathbf{x}) = \pm \mathbf{y}^S, \tag{4}$$

where $H^S : X \to \{+1, -1\}$ is a classifier that recognizes the presence (+1) or absence (-1) of a group of labels S in the data.

We propose a multi-class margin for evaluating the answer given by $\mathbf{f}^S(\mathbf{x})$. Data labels always belong to Y but predicted ones, $\mathbf{f}^S(\mathbf{x})$, belong to $\hat{Y}$. In consequence, depending on $s = |S|$, we have four possible margin values

$$z = \mathbf{y}^\top \mathbf{f}^S(\mathbf{x}) = \begin{cases} \pm \frac{K}{s(K-1)} & \text{if data label belongs to } S \\ \pm \frac{K}{(K-s)(K-1)} & \text{in another case} \end{cases} \tag{5}$$

where the sign is positive/negative if the partial classification is correct/incorrect. Derivations of the above expressions are in the Appendix.

We use an exponential loss to evaluate the margins in (5)

$$\mathcal{L}\left(\mathbf{y}, \mathbf{f}^S(\mathbf{x})\right) = \exp\left(\frac{-\mathbf{y}^\top \mathbf{f}^S(\mathbf{x})}{K}\right). \tag{6}$$

In consequence, the above vectorial codification of class labels with the exponential loss will produce different degrees of punishes and rewards depending on the number of classes separated by the weak-learner. Suppose that we fix a set of classes S and an associated weak-learner that separates S from the rest, $\mathbf{f}^S(\mathbf{x})$. We may also assume that $|S| \leq K/2$, since if $|S| > K/2$ we can choose $S' = S^c$ and in this case $|S'| \leq K/2$. The failure or success of $\mathbf{f}^S(\mathbf{x})$ in classifying an instance $\mathbf{x}$ with label $l \in S$ will have a larger margin than when classifying an instance with label $l \in S^c$. The margins in (5) provide the following rewards and punishes when used in conjunction with the exponential loss (6)

$$\mathcal{L}\left(\mathbf{y}, \mathbf{f}^S(\mathbf{x})\right) = \begin{cases} \exp\left(\frac{\mp 1}{s(K-1)}\right) & \text{if } \mathbf{y} \in S \\ \exp\left(\frac{\mp 1}{(K-s)(K-1)}\right) & \text{if not.} \end{cases} \tag{7}$$

In dealing with the class imbalance problem, the losses produced in (7) reflect the fact that the importance of instances in S is higher than those in S^c, since S is the smaller set. Hence, the cost of miss-classifying an instance in S outweighs that of classifying one in S^c [16]. This fact may also be intuitively interpreted in terms of the "difficulty" or amount of information provided by a classification. Classifying a sample in S provides more information, or, following the usual intuition behind boosting, is more "difficult", than the classification of an instance in S^c, since S^c is larger than S. The smaller the set S the more "difficult" or informative will be the result of the classification of an instance in it.

We can further illustrate this idea with an example. Suppose that we work on a classification problem with $K = 5$ classes. We may select $S_1 = \{1\}$ and $S_2 = \{1, 2\}$ as two possible sets of labels to be learned by our weak-learners. Samples in S_1 should be the more important than those in S_1^C or in S_2, since S_1 has the smallest class cardinality. Similarly, in general, it is easier to recognize data in S_2 than in S_1, since the latter is smaller; i.e. classifying a sample in S_1 provides more information than in S_2. Encoding labels with vectors from Y we will have the following margin values and losses

$$z = \mathbf{y}^\top \mathbf{f}^{S_1}(\mathbf{x}) = \begin{cases} \pm 5/4 \\ \pm 5/16 \end{cases} \Rightarrow \mathcal{L}(\mathbf{y}, \mathbf{f}^{S_1}) = \begin{cases} e^{\pm 1/4} = \{0.77, 1.28\} & \mathbf{y} \in S_1 \\ e^{\pm 1/16} = \{0.93, 1.06\} & \mathbf{y} \in S_1^c \end{cases}$$

$$z = \mathbf{y}^\top \mathbf{f}^{S_2}(\mathbf{x}) = \begin{cases} \pm 5/8 \\ \pm 5/12 \end{cases} \Rightarrow \mathcal{L}(\mathbf{y}, \mathbf{f}^{S_2}) = \begin{cases} e^{\pm 1/8} = \{0.88, 1.13\} & \mathbf{y} \in S_2 \\ e^{\pm 1/12} = \{0.92, 1.08\} & \mathbf{y} \in S_2^c \end{cases}$$

Everything we say about instances in S_1 will be the most rewarded or penalized in the problem, since S_1 is the smallest class set. Set S_2 is the

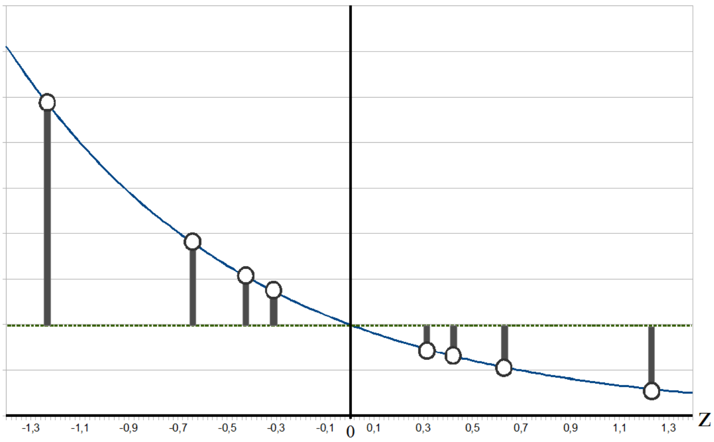

Figure 1: Values of the Exponential Loss Function over margins, z, for a classification problem with 5-classes. Possible margin values are obtained taking into account the expression (7) for $s = 1$ and $s = 2$.

second smallest, in consequence classification in that set will produce the second largest rewards and penalties. Similarly, we "say more" excluding an instance from $S_2 = \{1, 2\}$ than from $S_1 = \{1\}$, since S_2^c is smaller than S_1^c. In consequence, rewards and penalties for samples classified in S_2^c will be slightly larger than those in S_1^c. In Fig. 1 we display the loss values for the separators associated to the sets S_1 and S_2.

4. Partially Informative Boosting

In this section we present the structure of PIBoost whose pseudo-code we show in Algorithm 2. At each Boosting iteration we fit as many weak-learners as groups of labels, $G \subset \mathcal{P}(L)$, are considered. In our experiments we have chosen two types of subsets $\{$*all single labels*$\}$ and $\{$*all single labels and all pairs of labels*$\}$. The aim of each weak-learner is to separate its associated labels from the rest and persevere in this task iteration after iteration. That is why we call them *separators*. A weight vector $\mathbf{W}^S$ is associated to the separator of set S.

For each set $S \in G$ PIBoost builds a stage-wise additive model [20] of the form $\mathbf{f}_m(\mathbf{x}) = \mathbf{f}_{m-1}(\mathbf{x}) + \beta_m \mathbf{g}_m(\mathbf{x})$ (where super-index ^{S} is omitted for ease of notation). In step **2** of the algorithm we estimate constant β and

Algorithm 2 : PIBoost

1: Initialize weight vectors $\omega_i^S = 1/N$; with $i = 1, \ldots, N$ and $S \in G \subset \mathcal{P}(L)$.

2: For $m = 1$ until the number of iterations M and for each $S \in G$:

a) Fit a binary classifier $T_m^S(\mathbf{x})$ over training data with respect to its corresponding ω^S. Translate $T_m^S(\mathbf{x})$ into $\mathbf{g}_m^S : X \to \hat{Y}$.

b) Compute 2 types of errors associated with $T_m^S(\mathbf{x})$

$$\epsilon 1_{S,m} = \sum_{l_i \in S} \omega_i^S I\left(l_i \notin T_m^S(\mathbf{x}_i)\right)$$

$$\epsilon 2_{S,m} = \sum_{l_i \notin S} \omega_i^S I\left(l_i \notin T_m^S(\mathbf{x}_i)\right)$$

c) Calculate R_m^S, the only real positive root of the polynomial $P_m^S(x)$ defined according to (8).

d) Calculate $\beta_m^S = s(K-s)(K-1)\log\left(R_m^S\right)$

e) Update weight vectors as follows:

- If $l_i \in S$ then $\omega_i^S = \omega_i^S \cdot \left(R_m^S\right)^{\pm(K-s)}$
- If $l_i \notin S$ then $\omega_i^S = \omega_i^S \cdot \left(R_m^S\right)^{\pm s}$,

where the sign depends on whether T_m^S has a failure/success on $\mathbf{x}_i$.

f) Re-normalize weight vectors.

3: Output Final Classifier: $C(\mathbf{x}) = \arg\max_k F_k(\mathbf{x})$,
where $\mathbf{F}(\mathbf{x}) = (F_1(\mathbf{x}), ..., F_K(\mathbf{x})) = \sum_{m=1}^{M} \sum_{S \in G} \beta_m^S \mathbf{g}_m^S(\mathbf{x})$.

function $\mathbf{g}_m(\mathbf{x})$ for each label and iteration. The following *Lemma* solves the problem of finding those parameters.

Lemma 1. *Given an additive model $\boldsymbol{f}_m(\boldsymbol{x}) = \boldsymbol{f}_{m-1}(\boldsymbol{x}) + \beta_m \boldsymbol{g}_m(\boldsymbol{x})$ associated to a set of labels, $S \in G$, the solution to*

$$(\beta_m, \boldsymbol{g}_m(\boldsymbol{x})) = \arg\min_{\beta, \boldsymbol{g}(\boldsymbol{x})} \sum_{i=1}^{N} \exp\left(\frac{-\mathbf{y}_i^\top \left(\boldsymbol{f}_{m-1}(\boldsymbol{x}_i) + \beta \boldsymbol{g}(\boldsymbol{x}_i)\right)}{K}\right)$$

is

- $\boldsymbol{g}_m = \arg\min_{\boldsymbol{g}(\boldsymbol{x})} \sum_{i=1}^{N} \omega_i \cdot I\left(\mathbf{y}_i^\top \boldsymbol{g}(\boldsymbol{x}_i) < 0\right)$
- $\beta_m = s(K-s)(K-1)\log R$,

where R is the only real positive root of the polynomial

$$P_m(x) = \epsilon 1(K-s)x^{2(K-s)} + s\epsilon 2 x^K - s(A_2 - \epsilon 2)x^{(K-2s)} - (K-s)(A_1 - \epsilon 1) \quad (8)$$

where $A_1 = \sum_{l_i \in S} \omega_i$, $A_2 = \sum_{l_i \notin S} \omega_i$, i.e. $A_1 + A_2 = 1$, $\boldsymbol{W}_{m-1} = \{\omega_i\}$ the weight vector of iteration m-1, and $\epsilon 1 = \sum_{l_i \in S} \omega_i I(\mathbf{y}_i^\top \mathbf{g}(\mathbf{x}_i) < 0)$, $\epsilon 2 = \sum_{l_i \notin S} \omega_i I(\mathbf{y}_i^\top \mathbf{g}(\boldsymbol{x}_i) < 0)$.

The demonstration of this result is in the Appendix.

This lemma justifies steps `2:b)`[1], `2:c)` and `2:d)` in Algorithm 2. In case of $\mathbf{y} \in S$, the update rule `2:e)` follows from

$$\begin{aligned}
\omega_i^S &= \omega_i^S \cdot \exp\left(\frac{-1}{K}\mathbf{y}_i^\top \beta \mathbf{f}^S(\mathbf{x}_i)\right) \\
&= \omega_i^S \cdot \exp\left(\frac{-1}{K} s(K-s)(K-1)\log\left(R_m^S\right)\frac{\pm K}{s(K-1)}\right) \\
&= \omega_i^S \cdot \exp\left(\mp(K-s)\log\left(R_m^S\right)\right) = \omega_i^S \cdot \left(R_m^S\right)^{\mp(K-s)}
\end{aligned}$$

The case $\mathbf{y} \notin S$ provides an analogous expression.

The shape of the final classifier is easy and intuitive to interpret. The vectorial function built during the process collects in each k-coordinate information that can be understood as a *degree of confidence* for classifying sample $\mathbf{x}$ into class k. The classification rule assigns the label with highest

[1] In expression $l_i \notin T_m^S(\mathbf{x})$, the set $T_m^S(\mathbf{x})$ must be understood as $\left\{T_m^S(\mathbf{x}) = +1\right\} \equiv S$ and $\left\{T_m^S(\mathbf{x}) = -1\right\} \equiv S^C$.

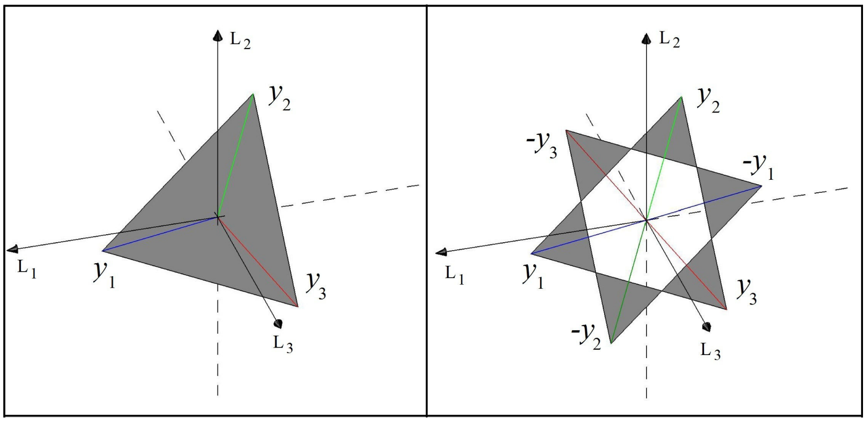

Figure 2: Margin vectors for a problem with three classes. Left figure presents the set of vectors Y. Right plot presents the set $\hat{Y}$.

value in its coordinate. This criterion has a geometrical interpretation provided by the codification of labels as K-dimensional vectors. Since the set $\hat{Y}$ contains margin vectors, the process of selecting the most probable one is carried out on the orthogonal hyperplane of $\mathbf{1} = (1, \ldots, 1)^\top$ (see Fig. 2). So, we build our decision on a subspace of $\mathbb{R}^K$ free of *total indifference* about labels. It means, that the final vector $\mathbf{F}(\mathbf{x})$ built during the process will usually present a dominant coordinate that represents the selected label. Ties between labels will only appear in degenerate cases. The plot on the right in Fig. 2 shows the set of pairs of vectors $\hat{Y}$ defined by our extension, whereas on the left are shown the set of vectors Y used in [18, 11]. Although the spanned gray hyperplane is the same, we exploit every binary answer in such a way that the negation of a class is directly translated into a new vector that provides positive evidence for the complementary set of classes in the final composition, $\mathbf{F}(\mathbf{x})$. The inner product of class labels $\mathbf{y} \in Y$ and classifier predictions, $\mathbf{f}(\mathbf{x}) \in \hat{Y}$, $\mathbf{y}^\top \mathbf{f}(\mathbf{x})$ produces a set of asymmetric margin values in such a way that, as described in section 3, all successes and failures do not have the same importance. Problems with four or more classes are more difficult to be shown graphically but allow richer sets of margin vectors.

The second key idea in PIBoost is that we can build a better classifier when collecting information from positive and negative classifications in $\hat{Y}$ than when using only the positive classifications in the set Y. Each weak-learner, or separator, $\mathbf{g}^S$, acts as a partial expert of the problem that provides us with a clue about *what is the label* of $\mathbf{x}$. Note here that when a weak-learner classifies $\mathbf{x}$ as belonging to a set of classes, the value of its associated step β, that depends on the success rate of the weak-learner, is evenly distributed among the classes in the set. In the same way, the bet will be used to evenly reduce the confidence on coordinates corresponding to non-selected classes. This balance inside selected and discarded classes is reflected in a margin value with a sensible multi-class interpretation. In other words, every answer obtained by a separator is directly translated into multi-class information in a fair way.

Reasoning in this way is a pattern of common sense. In fact we apply this philosophy in our everyday life when we try to guess something discarding possibilities. For instance, suppose that a boy knows that his favorite pen has been stolen in his classroom. He will ask each classmate what he knows about the issue. Perhaps doing this he will collect a pool of useful answers of the kind: "I think it was Jimmy", "I am sure it was not a girl", "I just know that it was not me nor Victoria", "I would suspect of Martin and his group of friends", etc. Combining all that information our protagonist should have one suspect. It's easy to find similarities between such a situation and the structure of PIBoost: the answer of each friend can be seen as a weak-learner T_t^S, the level of credibility (or trust) associated to each is our β_t, while the iteration value t can be thought as a measure of time in the relationship with the classmates.

4.1. AdaBoost as a special case of PIBoost

At this point we can verify that PIBoost applied to a two-class problem is equivalent to AdaBoost. In this case we only need to fit one classifier at each iteration[2]. Thus there will be only one weight vector to be updated and only one group of β constants.

It is also easy to match the expression of parameter β computed in PIBoost with the value of α computed in AdaBoost just by realizing that, fixed an iteration whose index we omit, the polynomial in step **2 - c)** is

$$P(x) = (\epsilon 1 + \epsilon 2)\, x^2 - (A_1 - \epsilon 1 + A_2 - \epsilon 2) = \epsilon \cdot x^2 - (1 - \epsilon)\,.$$

Solving this expression we get $R = \left(\frac{1-\epsilon}{\epsilon}\right)^{1/2}$, thus $\beta = \frac{1}{2} \log\left(\frac{1-\epsilon}{\epsilon}\right)$. What indeed is the value of α in AdaBoost.

Finally, it is immediate to see that the final classifiers are equivalent. If we transform AdaBoost's labels, $L = \{+1, -1\}$, into PIBoost's, $L' =$

[2] Separating the first class from the second is equivalent to separating the second from the first and, of course, there are no more possibilities.

$\{1, 2\}$, we get that $H(\mathbf{x}) = sign\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right)$ turns into $C(\mathbf{x}) = \arg\max_k F_k(\mathbf{x})$, where $F(x) = (F_1(\mathbf{x}), F_2(\mathbf{x})) = \sum_{m=1}^{M} \beta_m \mathbf{f}_m(\mathbf{x})$.

5. Experiments

Our goal in this section is to evaluate and compare the performance of PIBoost. We have selected fourteen data-sets from the UCI repository: *CarEvaluation*, *Chess*, *CNAE9*, *Isolet*, *Multifeatures*, *Nursery*,*OptDigits*, *PageBlocks*, *PenDigits*, *SatImage*, *Segmentation*, *Vehicle*, *Vowel* and *WaveForm*. They have different numbers of input variables (6 to 856), classes (3 to 26) and instances (846 to 28.056), and represent a wide spectrum of types of problems. Although some data-sets have separate training and test sets, we use both of them together, so the performance for each algorithm can be evaluated using cross-validation. Table 1 shows a summary of the main features of the databases.

Data-set	**Variables**	**Classes**	**Instances**
CarEvaluation	6	4	1728
Chess	6	18	28056
CNAE9	856	9	1080
Isolet	617	26	7797
Multifeatures	649	10	2000
Nursery	8	5	12960
OptDigits	64	10	5620
PageBlocks	10	5	5473
PenDigits	16	10	10992
SatImage	36	7	6435
Segmentation	19	7	2310
Vehicle	18	4	846
Vowel	10	11	990
Waveform	21	3	5000

Table 1: Summary of selected UCI data-sets

For comparison purposes we have selected three well-known multi-class Boosting algorithms. AdaBoost.MH [12] is perhaps the most prominent example of multi-class classifier with binary weak-learners. Similarly, SAMME [11] is a well-known representative of multi-class algorithms with multi-class weak-learners. Finally, multi-class GentleBoost [10] is an accurate method that treats labels separately at each iteration.

Selecting a weak-learner that provides a fair comparison among different Boosting algorithms is important at this point. SAMME requires multi-class weak-learners while, on the other hand, AdaBoost.MH and PIBoost can use even simple stump-like classifiers. Besides, multi-class GentleBoost requires the use of regression over continuous variables for computing its weak-learners. We chose classification trees as weak-learners, since they can be used in the first three algorithms, and regression trees for the last one.

For classification trees the following growing schedule was adopted. Each tree grows splitting impure nodes that present more than M/K-instances (where M is the number of samples selected for fitting the tree), so this value is taken as a lower bound for splitting. We found good results for the sample size parameter when $M < 0.4 \cdot N$, where N is the training data size. In particular we fix $M = 0.1 \cdot N$ for all data-sets. In the case of regression trees the growing pattern is similar but the bound of M/K-instances for splitting produced poor results. Here more complex trees achieve better performance. In particular when the minimum bound for splitting is $M/2K$-instances we got lower-enough error rates. A pruning process is carried out too in both types of trees.

We have experimented with two variants of PIBoost. The first one takes $G = \{All\ single\ labels\} \subset \mathcal{P}(L)$ as group of sets to separate while the second one, more complex, takes $G' = \{All\ single\ labels\} \cup \{All\ pairs\ of\ labels\}$. We must emphasize the importance of selecting a good group of separators in achieving the best performance. Depending on the number of classes, selecting an appropriate set G is a problem in itself. Knowledge of the dependencies among labels set will certainly help in designing a good set of separators. This is a problem that we do not address in this paper.

For the experiments we have fixed a number of iterations that depends on the algorithm and the number of labels of each data-set. Since the five algorithms considered in this section fit a different number of weak-learners at each iteration, we have selected the number of iterations of each algorithm so that all experiments have the same number of weak-learners (see Table 2). Remember that, when a data-set presents K-labels, PIBoost(2) fits $\binom{K}{2} + K$ separators per iteration while PIBoost(1) and GentleBoost fit only K. Besides SAMME and AdaBoost.MH fit one weak-learner per iteration. In Fig. 3 we plot the performance of all five algorithms.

The performance of a classifier corresponds to that achieved at the last iteration, combining all learned weak-learners. We evaluate the performance of the algorithms using 5-fold cross-validation. Table 3 shows these val-

Data-set	GentleBoost	AdaBoost.MH	SAMME	PIBoost(1)	PIBoost(2)	#WL
CarEvaluation (4)	70	280	280	70	40 [7]	280
Chess (18)	95	1710	1710	95	10 [171]	1710
CNAE9 (9)	100	900	900	100	20 [45]	900
Isolet (26)	135	3510	3510	135	10 [351]	3510
Multifeatures (10)	110	1100	1100	110	20 [55]	1100
Nursery (5)	120	600	600	120	40 [15]	600
OptDigits (10)	110	1100	1100	110	20 [55]	1100
PageBlocks (5)	120	600	600	120	40 [15]	600
PenDigits (10)	110	1100	1100	110	20 [55]	1100
SatImage (7)	80	560	560	80	20 [28]	560
Segmentation (7)	80	560	560	80	20 [28]	560
Vehicle (4)	70	280	280	70	40 [7]	280
Vowel (11)	120	1320	1320	120	20 [66]	1320
Waveform (3)	40	120	120	40	40 [3]	120

Table 2: Number of iterations considered for each Boosting algorithm. The first column displays the database name with the number of classes in parenthesis. Columns two to six display the number of iterations of each algorithm. For PIBoost(2) the number of separators per iteration appears inside brackets. The last column explicitly displays the number of weak-learners used for each database.

ues and their standard deviations. As can be seen, PIBoost (with its two variants) outperforms the rest of methods in many data-sets. Once the algorithms have been ranked by accuracy we used the Friedman test to asses whether the performance differences are statistically significant [22]. As was expected the null hypothesis (all algorithms have the same quality) is rejected with a p-value < 0.01. A post-hoc analysis was carried out too. We used the Nemenyi test to group the algorithms that present insignificant difference [22]. Figure 4 shows the result of the test for both $\alpha = 0.05$ and $\alpha = 0.1$ significance level. Summarizing, PIBoost(1) can be considered as good as PIBoost(2) and also as good as the rest of algorithms, but PIBoost(2) is significantly better than the latter. In addition, we used the Wilcoxon matched-pairs signed-ranks test to asses the statistical significance of the performance comparisons between pairs of algorithms [22]. Table 4 presents the p-values obtained after comparing PIBoost(1) and PIBoost(2) with the others. Again, it is clear that the latter is significantly better than the rest.

Additionally, we have also performed one more experiment with the *Amazon* database to asses the performance of PIBoost in a problem with a very high dimensional space and with a large number of classes. This database also belongs to the UCI repository. It has 1.500 sample instances

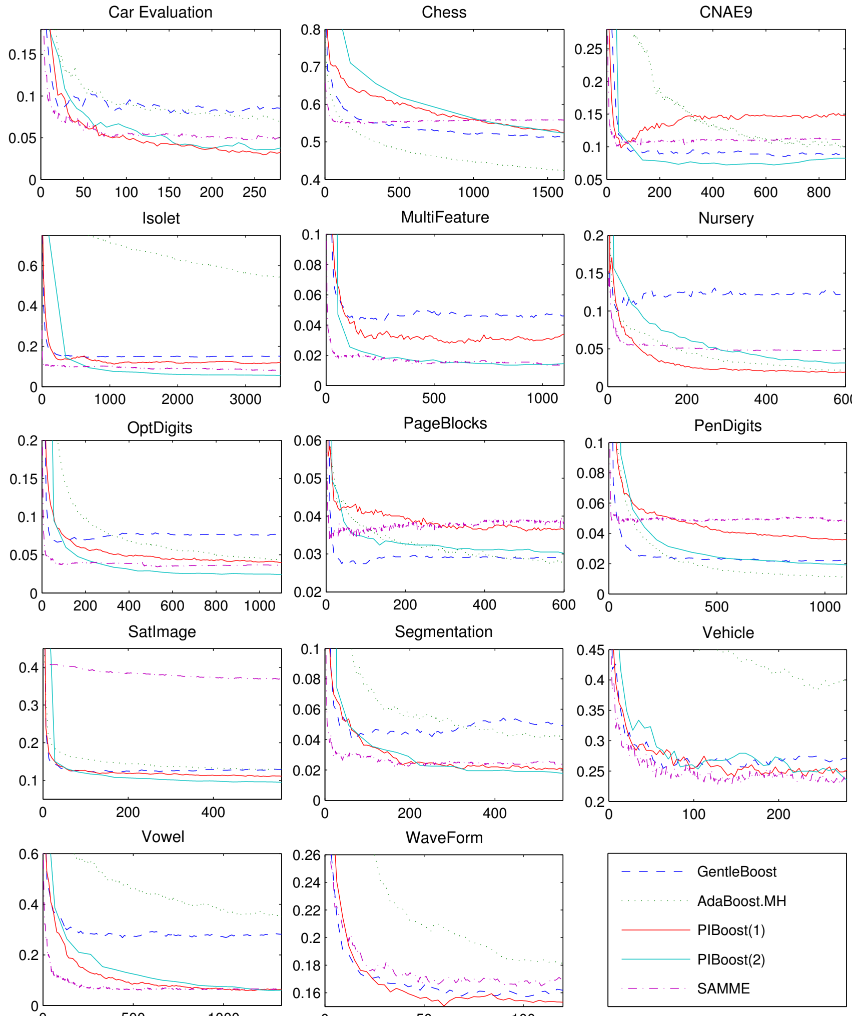

Figure 3: Plots comparing the performances of Boosting algorithms. In the vertical axis we display the error rate. In the horizontal axis we display the number of weak-learners fitted for each algorithm.

with 10.000 features grouped in 50 classes. With this database we followed the same experimental design as with the other databases, but only used the PIBoost(1) algorithm. In Figure 5 we plot the evolution in the performance of each algorithm as the number of weak learners increases. At the last iteration, PIBoost(1) had respectively an error rate and a standard deviation of 0.4213 and ($\pm 374 \times 10^{-4}$), whereas GentleBoost had 0.5107

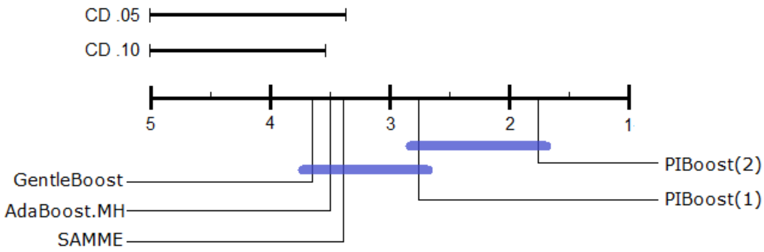

Figure 4: Diagram of the Nemenyi test. The average rank for each method is marked on the segment. We show critical differences for both $\alpha = 0.05$ and $\alpha = 0.1$ significance level at the top. We group with thick blue line algorithms with no significantly different performance.

Data-set	**GentleBoost**	**AdaBoost.MH**	**SAMME**	**PIBoost(1)**	**PIBoost(2)**
CarEvaluation	0.0852 (±121)	0.0713 (±168)	0.0487 (±111)	**0.0325** (±74)	0.0377 (±59)
Chess	0.5136 (±61)	**0.4240** (±34)	0.5576 (±63)	0.5260 (±118)	0.5187 (±74)
CNAE9	0.0870 (±239)	0.1028 (±184)	0.1111 (±77)	0.1472 (±193)	**0.0824** (±171)
Isolet	0.1507 (±94)	0.5433 (±179)	0.0812 (±185)	0.1211 (±253)	**0.0559** (±55)
Multifeatures	0.0460 (±128)	0.3670 (±822)	**0.0135** (±44)	0.0340 (±96)	0.0145 (±82)
Nursery	0.1216 (±60)	0.0203 (±32)	0.0482 (±58)	**0.0192** (±29)	0.0313 (±62)
OptDigits	0.0756 (±74)	0.0432 (±59)	0.0365 (±55)	0.0400 (±13)	**0.0240** (±41)
PageBlocks	0.0291 (±52)	**0.0276** (±46)	0.0386 (±87)	0.0364 (±47)	0.0302 (±50)
PenDigits	0.0221 (±11)	**0.0113** (±29)	0.0484 (±62)	0.0358 (±40)	0.0192 (±25)
SatImage	0.1294 (±32)	0.1318 (±51)	0.3691 (±120)	0.1113 (±62)	**0.0949** (±53)
Segmentation	0.0494 (±64)	0.0407 (±88)	0.0238 (±55)	0.0208 (±52)	**0.0177** (±61)
Vehicle	0.2710 (±403)	0.3976 (±297)	**0.2320** (±221)	0.2509 (±305)	0.2355 (±258)
Vowel	0.2818 (±322)	0.3525 (±324)	0.0667 (±114)	0.0646 (±183)	**0.0606** (±160)
Waveform	0.1618 (±75)	0.1810 (±72)	0.1710 (±109)	**0.1532** (±44)	**0.1532** (±44)

Table 3: Error rates of GentleBoost, AdaBoost.MH, SAMME, PIBoost(1) and PIBoost(2) algorithms for each data-set in table 1. Standard deviations appear inside parentheses in 10^{-4} scale. Bold values represent the best result achieved for each database.

and $(\pm 337 \times 10^{-4})$, SAMME 0.6267 and $(\pm 215 \times 10^{-4})$ and, finally, AdaBoost.MH 0.7908 and $(\pm 118 \times 10^{-4})$.

The experimental results confirm our initial intuition that by increasing the range of margin values and considering the asymmetries in the class

	GentleBoost	**AdaBoost.MH**	**SAMME**	**PIBoost(1)**
PIBoost(2)	0.0012	0.0203	0.0006	0.0081
PIBoost(1)	0.0580	0.1353	0.7148	

Table 4: P-values corresponding to Wilcoxon matched-pairs signed-ranks test.

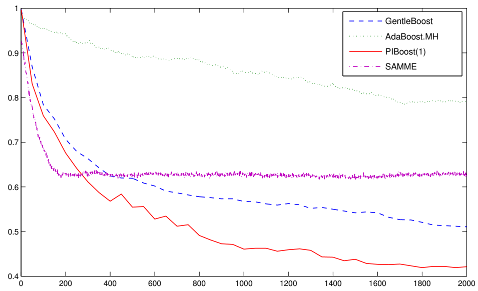

Figure 5: Plot comparing the performances of Boosting algorithms for the *Amazon* database. In the vertical axis we display the error rate. In the horizontal axis we display the number of weak-learners fitted for each algorithm.

distribution generated by the weak-learners we can significantly improve the performance of boosting algorithms. This is particularly evident in problems with a large number of classes and few training instances or those in a high dimensional space.

6. Related Work

In this section we relate our work with previous multi-class boosting algorithms. Recent results have addressed the problem of cost-sensitive or asymmetric boosting in the binary case [16, 23, 24]. In subsection 6.1 we will review these works and relate our multi-class solution to those results. Also, our approach like [13, 12, 14, 15], uses binary weak-learners to separate groups of classes. We will review multi-class boosting approaches with binary weak-learners in subsection 6.2. Moreover, our multi-class labels and weak-learner responses use a vectorial codification with a margin vector interpretation, like [18, 10, 11]. In subsection 6.3 we review boosting algorithms based on vectorial encodings and margin vectors.

6.1. *Asymmetric treatment of partial information*

The problem of learning from imbalanced data is concerned with the design of learning algorithms in the presence of underrepresented data and severe class distribution skews [17]. In the context of boosting, solutions to the class imbalance problem can be categorized as data level and algorithm level approaches. The goal at the data level is to re-weight or re-sample the data space so as to re-balance the class distribution. Approaches based on random oversampling [25] as well as random [26] and evolutionary [27] undersampling have been proposed. Alternatively, AdaBoost may also become an asymmetric boosting algorithm by changing only the initial data weights [24]. At the algorithm level, solutions try to adapt existing approaches to bias towards the small class [16] or to derive new cost-sensitive losses that produce asymmetric boosting algorithms [16, 23].

Our codification of class labels and classifier responses produces different margin values. This asymmetry in evaluating successes and failures in the classification may also be interpreted as a form of asymmetric boosting [16, 23]. As such it is related to the Cost-Sensitive AdaBoost in [23].

Using the cost matrix defined in Table 5, we can relate the PIBoost algorithm with the Cost-Sensitive AdaBoost [23]. If we denote $b \equiv \epsilon 1_S$, $d \equiv \epsilon 2_S$, $T_+ \equiv A_1$, $T_- \equiv A_2$ then the polynomial (8), $P^S(x)$, solved at each PIBoost iteration to compute the optimal step, β_m, along the direction of largest descent $\mathbf{g}_m(\mathbf{x})$ is equivalent to the following $\cosh(x)$-depending expression used in the *Cost-Sensitive AdaBoost* to estimate the same parameter [23]

$$2C_1 \cdot b \cdot \cosh(C_1\alpha) + 2C_2 \cdot d \cdot \cosh(C_2\alpha) = C_1 \cdot T_+ \cdot e^{-C_1\alpha} + C_2 \cdot T_- \cdot e^{-C_2\alpha},$$

where the costs $\{C_1, C_2\}$ are the non-zero values in Table 5.

Real\ Predicted	S	S^c
S	0	$\frac{1}{s(K-1)}$
S^c	$\frac{1}{(K-1)(K-s)}$	0

Table 5: Cost Matrix associated to a PIBoost's separator of a set S with $s = |S|$ classes.

In consequence, PIBoost is a boosting algorithm that combines a set of cost-sensitive binary weak-learners whose costs depend on the number of classes separated by each weak-learner.

6.2. *Boosting algorithms based on binary weak-learners*

A widely used strategy in machine learning for solving a multi-class classification problem with a binary classifier is to employ the *one-vs-all* method, that separates each class from the rest [28]. In the boosting literature, Shapire and Singer's AdaBoost.MH algorithm [12] is a prominent example of this approach. It creates a set of binary problems for each sample and each possible label. This algorithm was initially conceived for solving multi-label problems. However, it has been extensively used for solving the multi-class problem, in which class labels are mutually exclusive. As shown in [18] for the SVM case, this approach could perform poorly if there is no dominating class.

Following Dietterich and Bakiri's error-correcting-output-codes (ECOC) strategy [29], an alternative approach is to reduce the multi-class problem to multiple R-binary ones using a codeword to represent each class label. So for the r-th task a weak-learner $H_r : \mathbf{X} \rightarrow \{+1, -1\}$ is generated. The presence/absence of a group of labels over an instance is coded by a column vector belonging to $\{+1, -1\}^K$ or $\{1, 0\}^K$, in both cases $+1$ indicates presence of the labels selected by $H_r(\mathbf{x})$. Based on this idea several Boosting algorithms have been proposed, AdaBoost.OC [13], AdaBoost.MO [12] and AdaBoost.ECC [14]. The ECOC approach has been succesfully applied to a wide range of applications, such as face verification [30], facial expression recognition [31] or feature extraction [32].

The loss function applied for updating weights in AdaBoost.OC uses a relaxed error measurement termed pseudo-loss. AdaBoost.MO and AdaBoost.ECC use an exponential loss function with non-vectorial arguments. In section 3 we have highlighted the importance of using a *pure* multi-class loss function for achieving different margin values, hence penalizing binary failures into a real multi-class context. With our particular treatment for binary sub-problems we extend AdaBoost in a more natural way, because PIBoost can be seen as a group of several binary AdaBoost *well tied* via the multi-class exponential loss function and where every partial answer is well suited for the original multi-class problem.

Finally, the resulting schedule of PIBoost is similar to the $\{\pm 1\}$-matrix of ECOC algorithms, except for the presence of fractions. At each iteration of PIBoost there is a block of $|G|$-response vectors that, grouped as columns, form a $K \times |G|$-matrix similar to $|G|$-weak learners of any ECOC-based algorithm. However, in our approach, fractions let us make an even distribution of evidence among the classes in a set, whereas in the ECOC philosophy every binary sub-problem has the same importance for the final count. Moreover, the binary approaches reported so far do not consider the possible data imbalances produced by the number of classes falling on each side of the binary weak-learners.

6.3. Boosting algorithms based on vectorial encoding

An alternative way of extending binary boosting algorithms to the multi-class case is by encoding class membership in a set of vector-valued class codes and using an appropriate loss function. This idea of introducing a vectorial codification for extending a binary classifier to the multi-class case was first introduced for SVMs by Lee, Lin and Wahba [18]. Later, Zou, Zhu and Hastie introduced a theoretical basis for margin vectors and Fisher-consistent loss functions [10]. With their theory we can build multi-class boosting methods that handle multi-class weak-learners by coding labels as vectors and generalizing the concept of binary margin to multi-class problems in the following way. Given a classification function expressed in terms of margin vectors $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_K(\mathbf{x}))$, with $\sum_{j=1}^{K} f_j(\mathbf{x}) = 0$, if the real label of $\mathbf{x}$ is l the multi-class margin is the coordinate $f_l(\mathbf{x})$. Hence, a binary loss functions may be used for evaluating a multi-class decision. Based on this generalization, they derived multi-class generalizations of *GentleBoost* [10] and a new multi-class boosting algorithm minimizing the logit risk, *AdaBoost.ML* [10].

Almost parallel to this work Zhu, Zhou, Rosset and Hastie proposed SAMME (Stage-wise Additive Modeling using a Multi-class Exponential loss function) algorithm [11]. As described in section 2, this algorithm uses a multi-class exponential loss for evaluating classifications encoded with margin vectors when real labels are encoded likewise. The resulting algorithm only differs from AdaBoost (see Algorithm 1 and Algorithm 3) in step `5:` that now is $\alpha_m = \log\left((1 - Err_m)/Err_m\right) + \log(K - 1)$ and step `9:` that becomes

$$H(\mathbf{x}) = argmax_k \sum_{m=1}^{M} \alpha_m I\left(T_m(\mathbf{x}) = k\right).$$

The GAMBLE (Gentle Adaptive Multi-class Boosting Learning) algorithm also uses a multi-class vectorial codification and exponential loss function with the same type of weak-learners and structure of GentleBoost. The resulting multi-class Boosting schedule is merged with an active learning methodology to scale up to large data-sets [19].

Multi-class weak-learners have more parameters than simple binary classifiers and, consequently, they are more difficult to train and have a higher risk of over-fitting. For these reasons, most popular multi-class boosting algorithms are based on binary weak-learners.

In our approach, we use a vectorial codification with a sum-to-zero margin vector, like [18, 19, 10, 11] to represent multi-class data labels. However, our binary weak-learners code their answers in an extended set of vector

Algorithm 3 : SAMME

```
Initialize the Weight Vector W with uniform distribution ω_i = 1/N , i = 1, . . . , N.
for m = 1 to M do
    Fit a multi-class classifier T_m(x) to the training data using weights W.
    Compute weighted error: Err_m = Σ_{i=1}^{N} ω_i I (T_m(x_i) ≠ y_i).
    Compute α_m = log ((1 − Err_m)/Err_m) + log(K − 1).
    Update weight vector ω_i ← ω_i · exp (α_m I (T_m(x_i) ≠ y_i)) , i = 1, . . . , N.
    Re-normalize W.
end for
Output Final Classifier: H(x) = argmax_k Σ_{m=1}^{M} α_m I (T_m(x) = k)
```

codes that model the uncertainty in the classifier response, producing a larger set of asymmetric margin values that depend on the number of classes separated by each weak-learner.

7. Conclusions

We have proposed a new multi-class boosting algorithm called PIBoost, that is a generalization of existing binary multi-class boosting algorithms when we consider the asymmetries arising in the class distributions generated by the binarization process.

The main contribution of our framework is the use of binary classifiers whose response is coded in a multi-class vector and evaluated under an exponential loss function. Data labels and classifier responses are coded in different vector domains in such a way that they produce a set of asymmetric margin values that depend on the distribution of classes separated by the weak-learner. In this way the boosting algorithm properly addresses possible class imbalances appearing in the problem binarization. The range of rewards and punishments provided by this multi-class loss function is also related to the amount of information provided by each weak-learner. The most informative weak-learners are those that classify samples in the smallest class set and, consequently, their sample weight rewards and penalties are the largest. The ensemble response is the weighted sum of the weak-learner vector responses. Here the codification produces a fair distribution of the vote or evidence among the classes in the group. The resulting algorithm maintains the essence of AdaBoost, that, in fact, is a special case of PIBoost when the number of classes is two. Furthermore, the way it translates partial information about the problem into multi-class knowledge let us think of our method as the most canonical extension of AdaBoost using binary information.

The experiments performed confirm that PIBoost significantly improves the performance of other well known multi-class classification algorithms. However, we do not claim that PIBoost is the best multi-class boosting algorithm in the literature. Rather, we claim that the multi-class margin expansion introduced in the paper improves existing binary multi-class classification approaches and open new research venues in margin-based multi-class classification. We plan to extend this result to multi-label, multi-dimensional and multi-class cost-sensitive classifiers.

Appendix

Proof of expression (5)

Suppose, without loss of generality, that we work with an instance $\mathbf{x}$ that belongs to the first class and we try to separate the set S of the first s-labels from the rest using $\mathbf{f}^S(\mathbf{x})$. Suppose also that there is success when classifying with that separator over the instance. In that case the value of the margin will be

$$\begin{aligned}\mathbf{y}^\top \mathbf{f}^S(\mathbf{x}) &= \left(1, \frac{-1}{K-1}, ..., \frac{-1}{K-1}\right)\left(\frac{1}{s}, ..., \frac{1}{s}, \frac{-1}{K-s}, ..., \frac{-1}{K-s}\right)^\top \\ &= \frac{1}{s} - \frac{s-1}{s(K-1)} + \frac{(K-s)}{(K-1)(K-s)} = \frac{K}{s(K-1)}.\end{aligned}$$

If the separator is wrong then $\mathbf{f}^S(x)$ would have opposite sign and therefore the result.

Besides, suppose now that the real label of the instance is the same but now we separate the last s-labels from the rest. Suppose also that this time $\mathbf{f}^S$ erroneously classifies the instance as belonging to those last labels. The value of the margin will be

$$\begin{aligned}\mathbf{y}^\top \mathbf{f}^S(\mathbf{x}) &= \left(1, \frac{-1}{K-1}, ..., \frac{-1}{K-1}\right)\left(\frac{-1}{K-s}, ..., \frac{-1}{K-s}, \frac{1}{s}, ..., \frac{1}{s}\right)^\top \\ &= \frac{-1}{K-s} + \frac{(K-s-1)}{(K-1)(K-s)} - \frac{s}{(K-1)s} = \frac{-K}{(K-1)(K-s)}.\end{aligned}$$

Again, the sign of the result would be opposite if $\mathbf{f}^S$ excludes $\mathbf{x}$ from the first label group.

Demonstration of the Lemma

Let us fix a subset of s labels, $s = |S|$, and suppose that we have fitted a separator $\mathbf{f}_m(\mathbf{x})$ (whose S-index we omit) as an additive model $\mathbf{f}_{m+1}(\mathbf{x}) = \mathbf{f}_m(\mathbf{x}) + \beta\mathbf{g}(\mathbf{x})$, in the m-th step. We fix a $\beta > 0$ and rewriting the expression to look for the best $\mathbf{g}(\mathbf{x})$:

$$\begin{aligned}
\sum_{i=1}^{N} \exp\left(\frac{-1}{K} \cdot \mathbf{y}_i^\top \left(\mathbf{f}_m(\mathbf{x}_i) + \beta \mathbf{g}(\mathbf{x}_i)\right)\right) &= \\
&= \sum_{i=1}^{N} \omega_i \cdot \exp\left(\frac{-1}{K} \cdot \beta \cdot \mathbf{y}_i^\top \mathbf{g}(\mathbf{x}_i)\right) \\
&= \sum_{l_i \in S} \omega_i \cdot \exp\left(\frac{\mp\beta}{s(K-1)}\right) + \sum_{l_i \notin S} \omega_i \cdot \exp\left(\frac{\mp\beta}{(K-s)(K-1)}\right) \\
&= \left(\sum_{l_i \in S} \omega_i\right) \exp\left(\frac{-\beta}{s(K-1)}\right) + \left[\exp\left(\frac{\beta}{s(K-1)}\right) - \exp\left(\frac{-\beta}{s(K-1)}\right)\right] \cdot \\
&\quad \cdot \sum_{l_i \in S} \omega_i \cdot I(\mathbf{y}_i^\top \mathbf{g}(\mathbf{x}_i) < 0) \; + \left(\sum_{l_i \notin S} \omega_i\right) \exp\left(\frac{-\beta}{(K-s)(K-1)}\right) + \\
&\quad + \left[\exp\left(\frac{\beta}{(K-s)(K-1)}\right) - \exp\left(\frac{-\beta}{(K-s)(K-1)}\right)\right] \sum_{l_i \notin S} \omega_i \cdot I(\mathbf{y}_i^\top \mathbf{g}(\mathbf{x}_i) < 0).
\end{aligned}$$

The last expression is a sum of four terms. As can be seen, the first and third are constants while the second and fourth are the ones that depend on $\mathbf{g}(\mathbf{x})$. The values in brackets are positive constants. We obtain an immediate solution minimizing

$$\sum_{l_i \in S} \omega_i \cdot I(\mathbf{y}_i^\top \mathbf{g}(\mathbf{x}_i) < 0) + \sum_{l_i \notin S} \omega_i \cdot I(\mathbf{y}_i^\top \mathbf{g}(\mathbf{x}_i) < 0) = \sum_{i=1}^{N} \omega_i \cdot I(\mathbf{y}_i^\top \mathbf{g}(\mathbf{x}_i) < 0).$$

We reach the same conclusion assuming $\beta < 0$. Hence the first point of the Lemma follows.

Now suppose known $\mathbf{g}(\mathbf{x})$ and its error ϵ over training data. That error can be decomposed into two parts:

$$\begin{aligned}
\epsilon = \sum_{i=1}^{N} \omega_i \cdot I(\mathbf{y}_i^\top \mathbf{g}(\mathbf{x}_i) < 0) &= \sum_{l_i \in S} \omega_i \cdot I(\mathbf{y}_i^\top \mathbf{g}(\mathbf{x}_i) < 0) \; + \sum_{l_i \notin S} \omega_i \cdot I(\mathbf{y}_i^\top \mathbf{g}(\mathbf{x}_i) < 0) \\
&= \epsilon 1 + \epsilon 2.
\end{aligned}$$

The above expression can be written now as

$$A_1 \exp\left(\frac{-\beta}{s(K-1)}\right) + \left[\exp\left(\frac{\beta}{s(K-1)}\right) - \exp\left(\frac{-\beta}{s(K-1)}\right)\right]\epsilon 1 + \\ + A_2 \exp\left(\frac{-\beta}{(K-s)(K-1)}\right) + \left[\exp\left(\frac{\beta}{(K-s)(K-1)}\right) - \exp\left(\frac{-\beta}{(K-s)(K-1)}\right)\right]\epsilon 2,$$

where $A_1 = \sum_{l_i \in S} \omega_i$ and $A_2 = \sum_{l_i \notin S} \omega_i$. It can be easily verified that the above expression is convex with respect to β. So deriving w.r.t. β, equating to zero and simplifying terms we get:

$$\frac{\epsilon 1}{s} \exp\left(\frac{\beta}{s(K-1)}\right) + \frac{\epsilon 2}{K-s} \exp\left(\frac{\beta}{(K-s)(K-1)}\right) = \\ = \frac{(A_1 - \epsilon 1)}{s} \exp\left(\frac{-\beta}{s(K-1)}\right) + \frac{(A_2 - \epsilon 2)}{K-s} \exp\left(\frac{-\beta}{(K-s)(K-1)}\right).$$

There is no direct procedure to solve β here. We propose the change of variable $\beta = s(K-s)(K-1)\log(x)$ with $x > 0$. This change transform the last equation into the polynomial

$$(K-s)\epsilon 1 \cdot x^{(K-s)} + s\epsilon 2 \cdot x^{s} - s(A_2 - \epsilon 2) \cdot x^{-s} - (K-s)(A_1 - \epsilon 1) \cdot x^{-(K-s)} = 0,$$

or, equivalently, multiplying by $x^{(K-s)}$

$$(K-s)\epsilon 1 \cdot x^{2(K-s)} + s\epsilon 2 \cdot x^{K} - s(A_2 - \epsilon 2) \cdot x^{(K-2s)} - (K-s)(A_1 - \epsilon 1) = 0.$$

According to Descartes' *Theorem of the Signs* the last polynomial has a single real positive root. We estimate it numerically and undo the change of variable. This is the second point of the Lemma. Note here that the root will be zero only if $A_1 = \epsilon 1$, what makes $\beta = -\infty$. This possibility must be considered explicitly in the implementation.

Acknowledgments

Antonio Fernández-Baldera was funded by FPI grant BES2009-024828. Luis Baumela was partially funded by the Cajal Blue Brain project.

References

[1] P. Viola, M. Jones, Robust real-time face detection, International Journal of Computer Vision 57 (2) (2004) 137–154.

[2] A. Opelt, A. Pinz, A. Zisserman, Learning an alphabet of shape and appearance for multi-class object detection, International Journal of Computer Vision 80 (1) (2008) 16–44.

[3] F. Yuan, A double mapping framework for extraction of shape-invariant features based on multi-scale partitions with AdaBoost for video smoke detection, Pattern Recognition 45 (12) (2012) 4326 – 4336.

[4] P. Negri, N. Goussies, P. Lotito, Detecting pedestrians on a movement feature space, Pattern Recognition 47 (1) (2014) 56 – 71.

[5] A. Opelt, A. Pinz, M. Fussenegger, P. Auer, Generic object recognition with boosting, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (3) (2006) 416–431.

[6] M. Villamizar, J. Andrade-Cetto, A. Sanfeliu, F. Moreno-Noguer, Bootstrapping boosted random ferns for discriminative and efficient object classification, Pattern Recognition 45 (9) (2012) 3141 – 3153.

[7] L. Liu, L. Shao, P. Rockett, Boosted key-frame selection and correlated pyramidal motion-feature representation for human action recognition, Pattern Recognition 46 (7) (2013) 1810 – 1818.

[8] Y. Freund, R. E. Schapire, A decision theoretic generalization of on-line learning and an application to boosting, Journal of Computer and System Sciences 55 (1997) 199–139.

[9] J. Friedman, T. Hastie, R. Tibshirani, Additive logistic regression: a statistical view of boosting, The Annals of Statistics 28 (2) (2000) 337–407.

[10] H. Zou, J. Zhu, T. Hastie, New multicategory boosting algorithms based on multicategory fisher-consistent losses, Annals of Applied Statistics 2 (2008) 1290–1306.

[11] J. Zhu, H. Zou, S. Rosset, T. Hastie, Multi-class AdaBoost, Statistics and Its Interface 2 (2009) 349–360.

[12] R. E. Schapire, Y. Singer, Improved boosting algorithms using confidence-rated predictions, Machine Learning 37 (1999) 297–336.

[13] R. E. Schapire, Using output codes to boost multiclass learning problems, in: Proceedings of the International Conference on Machine Learning, 1997, pp. 313–321.

[14] V. Guruswami, A. Sahai, Multiclass learning, boosting and error correcting codes, in: Proceedings Conference on Learning Theory, 1999, pp. 145–155.

[15] E. L. Allwein, R. E. Schapire, Y. Singer, Reducing multiclass to binary: A unifying approach for margin classifiers, Journal of Machine Learning Research 1 (2000) 113–141.

[16] Y. Sun, M. S. Kamel, A. K. C. Wong, Y. Wang, Cost-sensitive boosting for classification of imbalanced data, Pattern Recognition 40 (12) (2007) 3358–3378.

[17] H. He, E. A. Garcia, Learning from imbalanced data, IEEE Transactions on Knowledge and Data Engineering 21 (9) (2009) 1263 –1284.

[18] Y. Lee, Y. Lin, G. Wahba, Multicategory support vector machines: theory and application to the classification of microarray data and satellite radiance data, Journal of the American Statistical Association 99 (2004) 67–81.

[19] J. Huang, S. Ertekin, Y. Song, H. Zha, C. L. Giles, Efficient multiclass boosting classification with active learning, in: SIAM International Conference on Data Mining, Society for Industrial and Applied Mathematics, 2007, pp. 1232–1243.

[20] T. Hastie, R. Tibshirani, Generalized Additive Models, Monographs on Statistics and Applied Probability, Chapman and Hall, 1990.

[21] H. Masnadi-Shirazi, N. Vasconcelos, V. Mahadevan, On the design of robust classifiers for computer vision, in: Proceedings International Conference on Computer Vision and Pattern Recognition, 2010, pp. 779–786.

[22] J. Demsar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.

[23] H. Masnadi-Shirazi, N. Vasconcelos, Cost-sensitive boosting, IEEE Transactions on Pattern Analysis and Machine Intelligence 33 (2011) 294–309.

[24] I. Landesa-Vázquez, J. L. Alba-Castro, Shedding light on the asymmetric learning capability of AdaBoost, Pattern Recognition Letters 33 (3) (2012) 247–255.

[25] N. V. Chawla, A. Lazarevic, L. O. Hall, K. W. Bowyer, Smoteboost: improving prediction of the minority class in boosting, in: Proceedings of the Principles of Knowledge Discovery in Databases, PKDD-2003, 2003, pp. 107–119.

[26] C. Seiffert, T. Khoshgoftaar, J. Van Hulse, A. Napolitano, Rusboost: A hybrid approach to alleviating class imbalance, IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 40 (1) (2010) 185–197.

[27] M. Galar, A. Fernndez, E. Barrenechea, F. Herrera, Eusboost: Enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling, Pattern Recognition 46 (12) (2013) 3460 – 3471.

[28] R. Rifkin, A. Klautau, In defense of one-vs-all classification, Journal of Machine Learning Research 5 (2004) 101–141.

[29] T. G. Dietterich, G. Bakiri, Solving multiclass learning problems via error-correcting output codes, Journal of Artificial Intelligence Research 2 (1) (1995) 263–286.

[30] N. Hatami, R. Ebrahimpour, R. Ghaderi, ECOC-based training of neural networks for face recognition, in: Proceedings Conference on Cybernetics and Intelligent Systems, 2008, pp. 450–454.

[31] R. S. Smith, T. Windeatt, Facial expression detection using filtered local binary pattern features with ECOC classifiers and platt scaling, in: Proceedings of the First Workshop on Applications of Pattern Analysis, Vol. 11 of JMLR Workshop and Conference Proceedings, 2010, pp. 111–118,.

[32] G. Zhong, C.-L. Liu, Error-correcting output codes based ensemble feature extraction, Pattern Recognition 46 (4) (2013) 1091 – 1100.