Researching on Combining Boosting Ensembles

Joaquín Torres-Sospedra Carlos Hernández-Espinosa Mercedes Fernández-Redondo

Abstract—As shown in the bibliography, training an ensemble of networks is an interesting way to improve the performance with respect to a single network. The two key factors to design an ensemble are how to train the individual networks and how to combine them to give a single output. Boosting is a well known methodology to build an ensemble. Some boosting methods use an specific combiner (Boosting Combiner) based on the accuracy of the network. Although the Boosting combiner provides good results on boosting ensembles, the simple combiner Output Average worked better in three new boosting methods we successfully proposed in previous papers. In this paper, we study the performance of sixteen different combination methods for ensembles previously trained with Adaptive Boosting and Average Boosting in order to see which combiner fits better on these ensembles. Finally, the results show that the accuracy of the ensembles trained with these original boosting methods can be improved by using the appropriate alternative combiner. In fact, the Output average and the Weighted average on low/medium sized ensembles provide the best results in most of the cases.

I. INTRODUCTION

One technique often used to increase the generalization capability with respect to a single neural network consists in training an ensemble of neural networks (Figure 1). This procedure consists of training a set of neural network with different weight initialization or properties in the training process and their combination in a suitable way.

The two key factors to design an ensemble are how to train the individual networks and how to combine the outputs provided by the networks to give a single output. Among the methods of training the individual networks and combining them there are an important number of alternatives. Our research group has performed some comparisons on methods to build [3], [6] and combine [14], [15] ensembles.

Reviewing the bibliography we can see that Adaptive Boosting (Adaboost) is one of the best performing methods to create an ensemble [4]. Adaboost is a method that construct a sequence of networks which overfits the training set used to train a neural network with hard to learn patterns. A sampling distribution is used to select the patterns we use to train the network. We can see the Adaboost diagram to train the networks in figure 2.

In previous papers, we successfully proposed three new boosting methods [16], [18], [17]. In those papers we noticed that in the majority of cases the Output average was better than the specific Boosting combiner.

Some authors like Breiman [1], Kuncheva [11] or Oza [13] have deeply studied and successfully improved Adaboost but any study on combining boosting methods has not been done.

In this paper, we present a comparison of sixteen different combiners on ensembles previously trained with Adaboost and Aveboost, two of the most important boosting methods, in order to test if the Boosting combiner is the most appropriate way to combine boosting ensembles.

This paper is organized as follows. Firstly, some theoretical concepts are briefly reviewed in section II. Then, the nine databases used and the experimental setup are described in section III. Finally, the experimental results and their discussion are in section IV.
A. Adaptive Boosting - AdaBoost

In AdaBoost, the successive networks are trained with a training data set \( T' \) selected at random from the original training data set \( T \), the probability of selecting a pattern from \( T \) is given by a sampling distribution associated to the network \( \text{Dist}_n \). The sampling distribution associated to a neural network is also based on the number of networks previously trained. The whole description is detailed in algorithm 1.

Algorithm 1 AdaBoost \( \{ T, V, k \} \)

- Initialize Sampling Distribution: \( \text{Dist}_1^x = 1/m \ \forall x \in T \)
  - for \( \text{net} = 1 \) to \( k \)
    - Create \( T' \) sampling from \( T \) using \( \text{Dist}_n^\text{net} \)
    - MF Network Training \( T', V \)
    - Calculate missclassified vector:
      \[
      \text{miss}_x^\text{net} = \begin{cases} 
        0 & \text{if } x \text{ is correctly classified} \\
        1 & \text{otherwise}
      \end{cases}
      \]
    - Calculate error:
      \[
      \epsilon_n = \sum_{x=1}^m \text{Dist}_n^\text{net} \cdot \text{miss}_x^\text{net}
      \]
    - Update sampling distribution:
      \[
      \text{Dist}_{n+1}^\text{net} = \text{Dist}_n^\text{net} \cdot \begin{cases} 
        \frac{1}{(2\epsilon_n)} & \text{if } \text{miss}_x^\text{net} \\
        \frac{1}{2(1-\epsilon_n)} & \text{otherwise}
      \end{cases}
      \]
  - end for

B. Averaged Boosting - Aveboost

Oza proposed in [13] Averaged Boosting (Algorithm 2). Aveboost is a method based on AdaBoost in which the sampling distribution related to a neural network is also based on the number of networks previously trained. The whole description is detailed in algorithm 2.

Algorithm 2 Aveboost \( \{ T, V, k \} \)

- Initialize Sampling Distribution: \( \text{Dist}_1^x = 1/m \ \forall x \in T \)
  - for \( \text{net} = 1 \) to \( k \)
    - Create \( T' \) sampling from \( T \) using \( \text{Dist}_n^\text{net} \)
    - MF Network Training \( T', V \)
    - Calculate missclassified vector:
      \[
      \text{miss}_x^\text{net} = \begin{cases} 
        0 & \text{if } x \text{ is correctly classified} \\
        1 & \text{otherwise}
      \end{cases}
      \]
    - Calculate error:
      \[
      \epsilon_n = \sum_{x=1}^m \text{Dist}_n^\text{net} \cdot \text{miss}_x^\text{net}
      \]
    - Update sampling distribution:
      \[
      C_x^\text{net} = \text{Dist}_n^\text{net} \cdot \begin{cases} 
        \frac{1}{2\epsilon_n} & \text{if } \text{miss}_x^\text{net} \\
        \frac{1}{2(1-\epsilon_n)} & \text{otherwise}
      \end{cases}
      \]
      \[
      \text{Dist}_{n+1}^\text{net} = \frac{\text{net} \cdot \text{Dist}_n^\text{net} + C_x^\text{net}}{\text{net}+1}
      \]
  - end for

AdaBoost and Aveboost use an specific combination method, Boosting combiner, to combine the networks and get the final output or hypothesis eq.1.

\[
h(x) = \arg \max_{c=1, \ldots, \text{classes}} \sum_{\text{net}: h^\text{net}(x) = c} \log \frac{1 - \epsilon_n}{\epsilon_n}
\]  

C. Alternative Combiners

In this subsection, we briefly review the alternative combiners we have used to obtain the experimental results.

1) Average: This approach simply averages the individual classifier outputs across the different classifiers. The output yielding the maximum of the averaged values is chosen as the correct class.

2) Majority Vote: Each classifier provides a vote to a class, given by the highest output. The correct class is the one most often voted by the classifiers.

3) Winner Takes All (WTA): In this method, the class with overall maximum output across all classifier and outputs is selected as the correct class.

4) Borda Count: For any class \( c \), the Borda count is the sum of the number of classes ranked below \( c \) by each classifier [7], [19]. If \( B_j(c) \) is the number of classes ranked below the class \( c \) by the \( j \)th classifier, then the Borda count for class \( c \) is in the following equation.

\[
B(c) = \sum_{n=1}^k B_{\text{net}}(c)
\]

5) Bayesian Combination: This combination method was proposed in reference [22]. According to this reference a belief value that the pattern \( x \) belongs to class \( c \) can be approximated by the following equation.

\[
\text{Bel}(c) = \frac{\prod_{n=1}^k \Pi_{i=1}^{\text{classes}} P(x \in q_i | \lambda_{\text{net}}(x) = j_{\text{net}})}{\sum_{i=1}^{\text{classes}} \prod_{n=1}^k P(x \in q_i | \lambda_{\text{net}}(x) = j_{\text{net}})}
\]

Where the conditional probability that sample \( x \) actually belongs to class \( c \), given that classifier \( \text{net} \) assign it to class \( j \) \( (\lambda_{\text{net}}(x) = j_{\text{net}}) \) can be estimated from the values of the confusion matrix [19].

6) Weighted Average: This method introduces weights to the outputs of the different networks prior to averaging. The weights try to minimize the difference between the output of the ensemble and the desired or true output. The weights can be estimated from the error correlation matrix. The full description of the method can be found in [10], [19].

7) Choquet Integral: This method is based in the fuzzy integral [2], [5] and the Choquet integral. The method is complex, its full description can be found in [19].

8) Fuzzy Integral with Data Dependent Densities: It is another method based on the fuzzy integral and the Choquet integral. But in this case, prior to the application of the method it is performed a partition of the input space into \( n \) regions by frequency sensitive learning algorithm (FSL). The full description can be found in reference [19].

9) Weighted Average with Data Dependent weights: This method is the weighted average described above. But in this case, a partition of the space is performed by FSL algorithm and the weights are calculated for each partition. We have a different combination scheme for the different partitions of the space. The method is fully described in [19].
10) **BADD Defuzzification Strategy:** It is another combination method based on fuzzy logic concepts. The method is complex and the description can also be found in [19].

11) **Zimmermann’s Compensatory Operator:** This combination method is based in the Zimmermann’s compensatory operator described in [23]. The method is complex and can be found in [19].

12) **Dynamically Averaged Networks:** Two versions of Dynamically Averaged Networks were proposed by Jimenez [8], [9]. In these methods instead of choosing static weights derived from the network output on a sample of the input space, we allow the weights to adjust to be proportional to the certainties of the respective network output.

13) **Nash Vote:** In this method each voter assigns a number between zero and one for each candidate output. The product of the voter’s values is compared for all candidates. The higher is the winner. The method is reviewed in reference [20].

14) **Stacked Combiners (Stacked and Stacked+):** The training in Stacked Generalization is divided into two steps. In the first one, the expert networks are trained. In the second one, the combination networks are trained with the outputs provided by the experts.

**Stacked Generalization** [21] can be adapted to combine ensembles of neural networks. In [15], Stacked and Stacked+, two combiners based on Stacked Generalization, were successfully proposed.

In those combiners, the networks of the ensembles were used as expert networks. Then, the outputs provided by the ensembles on the original training set were used to train the combination networks. In stacked, the combination networks use the information provided by the networks along with the original class label (output) in order to get a more accurate classification system. In Stacked+, the combination networks use the information provided by the networks, the original class label and the original input data in order to improve the performance of the ensemble.

In the experiments carried out in this paper, only one single combination network has been applied to combine each ensemble.

### III. Experimental Setup

In this section, we describe the experimental setup, the datasets we have used in our experiments.

In our experiments we have used ensembles of 3, 9, 20 and 40 Multilayer Feedforward networks we previously trained with Adaptive Boosting and Averaged Boosting on the databases described in subsection III-A using the training parameters described in table I. Then we have applied the Boosting combiner, as described in the original Boosting methods, and the alternative combiners to get the results.

Moreover, we have repeated the whole learning process 10 times using different training, validation and test sets. With this procedure we can obtain a mean performance of the ensemble for each database and an error in the performance calculated by standard error theory.

### A. Datasets

We have used the following nine classification problems from the UCI repository of machine learning databases [12]: Balance Scale Database (bala), Cylinder Bands Database (band), BUPA liver disorders (bupa), Australian Credit Approval (cred), Glass Identification Database (glas), Heart Disease Databases (heart), The Monk’s Problem 1 and 2 (mok1 and mok2) and Congressional Voting Records Database (vote).

<table>
<thead>
<tr>
<th>database</th>
<th>hidden</th>
<th>step</th>
<th>mom</th>
<th>ite</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>bala</td>
<td>20</td>
<td>0.1</td>
<td>0.05</td>
<td>5000</td>
<td>87.6 ± 0.6</td>
</tr>
<tr>
<td>band</td>
<td>23</td>
<td>0.1</td>
<td>0.05</td>
<td>5000</td>
<td>72.4 ± 1.0</td>
</tr>
<tr>
<td>bupa</td>
<td>11</td>
<td>0.1</td>
<td>0.05</td>
<td>8500</td>
<td>58.3 ± 0.6</td>
</tr>
<tr>
<td>cred</td>
<td>15</td>
<td>0.1</td>
<td>0.05</td>
<td>8500</td>
<td>85.6 ± 0.5</td>
</tr>
<tr>
<td>glas</td>
<td>3</td>
<td>0.1</td>
<td>0.05</td>
<td>4000</td>
<td>78.5 ± 0.9</td>
</tr>
<tr>
<td>hear</td>
<td>2</td>
<td>0.1</td>
<td>0.05</td>
<td>5000</td>
<td>82.0 ± 0.9</td>
</tr>
<tr>
<td>mok1</td>
<td>0</td>
<td>0.1</td>
<td>0.05</td>
<td>3000</td>
<td>74.3 ± 1.1</td>
</tr>
<tr>
<td>mok2</td>
<td>20</td>
<td>0.1</td>
<td>0.05</td>
<td>7000</td>
<td>65.9 ± 0.5</td>
</tr>
<tr>
<td>vote</td>
<td>1</td>
<td>0.1</td>
<td>0.05</td>
<td>2500</td>
<td>59.0 ± 0.4</td>
</tr>
</tbody>
</table>

### Table II

**Training parameters - Stacked and Stacked+ on AdaBoost**

<table>
<thead>
<tr>
<th>database</th>
<th>ite</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>bala</td>
<td>7500</td>
<td></td>
</tr>
<tr>
<td>band</td>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>bupa</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>cred</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>hear</td>
<td>7000</td>
<td></td>
</tr>
<tr>
<td>mok1</td>
<td>5000</td>
<td></td>
</tr>
<tr>
<td>mok2</td>
<td>5000</td>
<td></td>
</tr>
<tr>
<td>vote</td>
<td>5000</td>
<td></td>
</tr>
</tbody>
</table>
The optimal parameters of the Multilayer Feedforward networks (Hidden units, Adaptation step, Momentum rate and Number of iterations) we have used to train the networks of the ensembles are shown in Table I.

The optimal parameters of the Multilayer Feedforward networks (Hidden units, Adaptation step, Momentum rate and Number of iterations) we have used to train the combination networks of combiners Stacked and Stacked+ are shown in Tables II (combining ensembles trained with Adaboost) and III (combining ensembles trained with Aveboost).

### Table II

<table>
<thead>
<tr>
<th>nets</th>
<th>h.u.</th>
<th>step</th>
<th>mom</th>
<th>ite</th>
<th>h.u.</th>
<th>step</th>
<th>mom</th>
<th>ite</th>
</tr>
</thead>
<tbody>
<tr>
<td>bala</td>
<td>3</td>
<td>9</td>
<td>0.40</td>
<td>0.05</td>
<td>25</td>
<td>0.10</td>
<td>0.20</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>2</td>
<td>0.10</td>
<td>0.05</td>
<td>7500</td>
<td>0.20</td>
<td>0.10</td>
<td>7500</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>2</td>
<td>0.05</td>
<td>0.01</td>
<td>6000</td>
<td>0.05</td>
<td>0.10</td>
<td>7500</td>
</tr>
<tr>
<td>band</td>
<td>3</td>
<td>9</td>
<td>0.40</td>
<td>0.05</td>
<td>6500</td>
<td>0.20</td>
<td>0.10</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>7</td>
<td>0.40</td>
<td>0.20</td>
<td>4000</td>
<td>0.40</td>
<td>0.01</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>12</td>
<td>0.40</td>
<td>0.10</td>
<td>3000</td>
<td>0.20</td>
<td>0.10</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>13</td>
<td>0.40</td>
<td>0.10</td>
<td>7500</td>
<td>0.20</td>
<td>0.10</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
<td>5000</td>
<td>0.20</td>
<td>0.05</td>
<td>7500</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>16</td>
<td>0.40</td>
<td>0.01</td>
<td>4000</td>
<td>0.40</td>
<td>0.01</td>
<td>1000</td>
</tr>
<tr>
<td>bupa</td>
<td>4</td>
<td>4</td>
<td>0.40</td>
<td>0.05</td>
<td>4000</td>
<td>0.40</td>
<td>0.01</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>0.40</td>
<td>0.05</td>
<td>7500</td>
<td>0.20</td>
<td>0.01</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>26</td>
<td>0.40</td>
<td>0.05</td>
<td>4000</td>
<td>0.20</td>
<td>0.01</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>23</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
<td>0.20</td>
<td>0.01</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>2</td>
<td>0.40</td>
<td>0.10</td>
<td>5000</td>
<td>0.20</td>
<td>0.20</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td>0.40</td>
<td>0.10</td>
<td>7500</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>14</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>25</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>21</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
</tr>
<tr>
<td>hear</td>
<td>3</td>
<td>10</td>
<td>0.40</td>
<td>0.20</td>
<td>6000</td>
<td>0.40</td>
<td>0.01</td>
<td>7500</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>5</td>
<td>0.40</td>
<td>0.20</td>
<td>1750</td>
<td>0.40</td>
<td>0.05</td>
<td>1500</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>25</td>
<td>0.40</td>
<td>0.20</td>
<td>3000</td>
<td>0.40</td>
<td>0.20</td>
<td>1500</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>7</td>
<td>0.20</td>
<td>0.05</td>
<td>4000</td>
<td>0.40</td>
<td>0.20</td>
<td>2500</td>
</tr>
<tr>
<td>mok1</td>
<td>3</td>
<td>29</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>28</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>29</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>30</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
<td>0.40</td>
<td>0.20</td>
<td>7500</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>15</td>
<td>0.20</td>
<td>0.10</td>
<td>5000</td>
<td>0.40</td>
<td>0.05</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>2</td>
<td>0.10</td>
<td>0.10</td>
<td>5000</td>
<td>0.40</td>
<td>0.01</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>2</td>
<td>0.05</td>
<td>0.01</td>
<td>5000</td>
<td>0.40</td>
<td>0.01</td>
<td>500</td>
</tr>
<tr>
<td>mok2</td>
<td>14</td>
<td>0.40</td>
<td>0.10</td>
<td>5000</td>
<td>0.40</td>
<td>0.10</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2</td>
<td>0.40</td>
<td>0.20</td>
<td>5000</td>
<td>0.40</td>
<td>0.10</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>30</td>
<td>0.40</td>
<td>0.10</td>
<td>5000</td>
<td>0.40</td>
<td>0.01</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>30</td>
<td>0.40</td>
<td>0.05</td>
<td>5000</td>
<td>0.40</td>
<td>0.01</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>25</td>
<td>0.40</td>
<td>0.20</td>
<td>5000</td>
<td>0.40</td>
<td>0.05</td>
<td>500</td>
</tr>
</tbody>
</table>

Finally, we set to $n = 5$ the numbers of regions used in the combiners based on data depend densities. The parameters have been set after an exhaustive trial and error procedure using the training and validation sets.

### IV. Results and Discussion

Due to the lack of space, the general results on combining ensembles trained with Adaboost and Aveboost are shown in this section instead of showing the complete results. The general measurements used in this paper are described in subsection IV-A.

#### A. General measurements

In our experiments, we have calculated the Increase of Performance (IoP) and the Percentage of Error Reduction (PER) of the results with respect to a single network in order to perform an exhaustive comparison.

The $IoP$ value is an absolute measurement that denotes the increase of performance of the ensemble with respect to a single network.

$$IoP = Error_{SingleNet} - Error_{Ensemble}$$

The $PER$ value is a relative measurement which ranges from 0%, where there is no improvement by the use of an ensemble method with respect to a single network, to 100%.

$$PER = 100 \cdot \frac{Error_{SingleNet} - Error_{Ensemble}}{Error_{SingleNet}}$$

A negative value on the IoP and on the PER means that the performance of the ensemble is worse than the performance of the single network.

Finally, we have calculated the mean Increase of Performance and the mean Percentage of Error Reduction across all databases to get a general measurement to compare the methods presented in the paper. The results on combining Adaboost are presented in subsection IV-B whereas the results on combining Aveboost are in subsection IV-C.

#### B. Adaboost results

In this subsection the results of the different combiners on ensembles trained with Adaptive Boosting are shown. Table IV shows the mean $IoP$ whereas table V shows the mean $PER$ for the original ensembles (combined with the Boosting combiner) and for the same ensembles combined with the alternative combiners described in subsection II-C.

Moreover, table VI shows the best performance for each database on ensembles trained with the original Adaboost (applying the Boosting combiner). The table also shows the best performance of these ensembles combined with the sixteen alternative combiners.

#### C. Aveboost results

In this subsection the results of the different combiners on ensembles trained with Averaged Boosting are shown. Table VII shows the mean $IoP$ whereas table VIII shows the mean $PER$ for the original ensembles (combined with the Boosting combiner) and for the same ensembles combined with the alternative combiners described in subsection II-C.

Moreover, table IX shows the best performance for each database on ensembles trained with the original Aveboost (applying the Boosting combiner). The table also shows the best performance of these ensembles combined with the sixteen alternative combiners.
D. Discussion

We see that the Boosting combiner is not the best alternative in Adaboost, in all the cases there is at least one combiner with we got better mean IoP and PER (tables IV-V). If we analyse table VI, we can see that the boosting combiner only provides the best result on databases bupa and vote.

We can also see that, in the majority of cases, the mean IoP and PER of the boosting combiner does not reach the best value in Aveboost. Moreover, the boosting combiner only provides the best result for database hear (table IX).

Moreover, we can notice that in some cases the accuracy is highly improved by applying an alternative combiner.
In this paper, we have performed a comparison among sixteen combiners on ensembles trained with Adaptive Boosting and Averaged Boosting. To carry out our comparison we have used ensembles of 3, 9, 20 and 40 networks previously trained with Adaptive boosting and Averaged boosting and the accuracy of the ensemble using the Boosting Combiner. Alternatively, we have applied sixteen different combiners on these ensembles to test if the boosting combiner is the best method to combine the networks of a boosting ensemble. Moreover, we also want to know which is the most appropriate combiner in each case. Finally, we have calculated the mean Increase of Performance and the mean Percentage of Error Reduction with respect to a single network to compare the combiners. Furthermore, the best accuracy for each database with the original methods, Boosting combiner on Adaboost and Aveboost, and applying the sixteen alternative combiners on these ensembles have been shown.

According the general results, the Boosting combiner is not the most effective way to combine an ensemble trained with Boosting. In a wide number of cases the original results have been improved with the use of an alternative combiner. In general, the Output average, the Weighted average and the Stacked combiners are the best combiners on ensembles trained with Adaboost. The Output average, the Weighted average with data depend densities and the Boosting combiner are the best combiners on ensembles trained with Aveboost.

According the best performance for each database (tables VI and IX), we can see that the Output average, the Weighted average and the Stacked combiners should be seriously considered for combining boosting ensembles because the Boosting combiner provides the best results only in 16.6% of the cases. In addition, in a 50% of the cases not only have the accuracy of the ensembles been improved with the use of an alternative combiner, the numbers of networks to get the best result is also reduced. For instance, the best accuracy for database hear using the original Adaboost (applying the Boosting combiner) was got with the 40-network ensembles (82.2 ± 1.8). The best overall accuracy for this database using Adaboost was got by applying the Weighted average with data depend densities to the 9-networks ensembles (84.6 ± 1.4). The accuracy was improved in 2.4%, the error rate was reduced in 0.2% and the number of networks required were reduced from 40 to 9.

Nowadays, we are extending the comparison we have performed using more databases and using more Boosting methods. The results we are getting also show that the Boosting combiner does not provide either the best general results (best mean IoP or PER) or the best performance for each database. Furthermore, we are working on an advanced combination method based on the boosting combiner we think could increase the accuracy of the Boosting ensembles.

We can conclude by remarking that the accuracy of a boosting ensemble can be improved and its size can be reduced by applying the Output average or advanced combiners like the Weighted average or the Stacked combiners.

References