Can AdaBoost.M1 Learn Incrementally? A Comparison to Learn^{++} Under Different Combination Rules

Hussein Syed Mohammed, James Leander, Matthew Marbach, and Robi Polikar*

Electrical and Computer Engineering, Rowan University, Glassboro,
NJ 08028, USA
polikar@rowan.edu

Abstract. We had previously introduced Learn^{++}, inspired in part by the ensemble based AdaBoost algorithm, for incrementally learning from new data, including new concept classes, without forgetting what had been previously learned. In this effort, we compare the incremental learning performance of Learn^{++} and AdaBoost under several combination schemes, including their native, weighted majority voting. We show on several databases that changing AdaBoost's distribution update rule from hypothesis based update to ensemble based update allows significantly more efficient incremental learning ability, regardless of the combination rule used to combine the classifiers.

1 Introduction

Learning from new data without forgetting prior knowledge is known as incremental learning, and it is encountered often real world applications. This is because sufficiently dense and a representative set of training examples is usually required for satisfactory classifier performance, however, acquisition of such a representative dataset often become available in small and separate batches at different times. Under such conditions, it is necessary to incrementally update an existing classifier to accommodate new data while retaining the information from old data.

Traditionally, when new data become available, previous classifiers are discarded and retrained with the composite data obtained by combining all the data accumulated thus far. However, this approach results in loss of all previously acquired information, and it is commonly known as *catastrophic forgetting* [1]. Furthermore, this approach may not even be feasible, if the original dataset is no longer available.

Learning new data incrementally without forgetting previously acquired knowledge raises the stability-plasticity dilemma [2]: acquiring new knowledge requires plasticity, whereas retaining previously acquired knowledge requires stability. The challenge is to achieve a meaningful balance between the two conflicting properties.

Various forms of incremental learning have been studied under various conditions. In one extreme end, incremental learning is trivialized by allowing retraining with old data, while on the other end, an incremental learning algorithm is expected to learn in an online incremental setting, where learning is carried out in an instance-by-instance

* Corresponding author.

basis with some instances introducing new classes. Algorithms that are currently available for incremental learning, such as ARTMAP [3], typically fall somewhere in the middle of this spectrum.

2 Learn^{++} for Incremental Learning

Learn^{++} inspired in part by the ensemble structure of the AdaBoost.M1 algorithm [4], exploits the synergistic expressive power of an ensemble of classifiers to incrementally learn additional information from new data [5,6]. Specifically, for each database that becomes available, Learn^{++} generates a number of diverse classifiers, which are then combined using a suitable combination rule (originally, the weighted majority voting). The pseudocode of Learn^{++} is shown in Figure 1.

Inputs to Learn^{++} are the training data $\boldsymbol{S_k}$ of m_k samples drawn from the current database $\boldsymbol{DB_k}$, a supervised learning algorithm **BaseClassifier,** and an integer T_k, specifying the number of classifiers to be generated for database $\boldsymbol{DB_k}$. Learn^{++} generates an ensemble of classifiers using different subsets of each training data, $\boldsymbol{S_k}$. This is achieved by iteratively updating a distribution $\boldsymbol{D_t}$, $t = 1,\dots,T_k$ from which training

```
Inputs: For each dataset drawn from $\boldsymbol{DB_k}$ $k=1,2,\dots,K$
  •  Sequence of $m_k$ examples $\boldsymbol{S_k}=\{(\mathbf{x}_i,y_i) \mid i=1,\dots,m_k\}$
  •  Supervised learning algorithm BaseClassifier.
  •  Integer $T_k$, specifying the number of iterations.
Do for each $k=1,2,\dots,K$:
    Initialize $w_1(i)=D_1(i)=1/m_k,\ \forall i,\ i=1,2,\cdots,m_k$                                  (1)
    If $k>1$, Go to Step 5, evaluate current ensemble on new $\boldsymbol{S_k}$, update weight distribution; End If
    Do for $t = 1,2,\dots,T_k$:
        1. Set $\boldsymbol{D_t} = \mathbf{w}_t \big/ \sum_{i=1}^{m_k} w_t(i)$ so that $\boldsymbol{D_t}$ is a distribution
           (2)
        2. Draw a training $\boldsymbol{TR_t}$ subset from the distribution $\boldsymbol{D_t}$, and train BaseClassifier with $\boldsymbol{TR_t}$.
        3. Obtain a hypothesis $h_t$ and calculate its error on $\boldsymbol{S_k}$. If $\varepsilon_t > ½$, discard $h_t$, go to step 2.
           $\varepsilon_t = \sum_{i:h_t(\mathbf{x_i})\neq y_i} D_t(i)$                                    (3)
        4. Call CombinationRule, and obtain the composite hypothesis $H_t$
        5. Compute the error of the composite hypothesis
           $E_t = \sum_{i:H_t(\mathbf{x}_i)\neq y_i} D_t(i) = \sum_{i=1}^{m_k} D_t(i)\big[\,|H_t(\mathbf{x}_i)\neq y_i|\,\big]$          (4)
        6. Set $B_t = E_t \big/ (1-E_t)$, and update the weights:                          (5)
           $w_{t+1}(i) = w_t(i)\times B_t^{1-[|H_t(\mathbf{x_i})\neq y_i|]} = w_t(i)\times\begin{cases} B_t, & if\ \ H_t(\mathbf{x_i})=y_i \\ 1\ , & otherwise \end{cases}$     (6)
Call CombinationRule and output the final hypothesis.
```

Fig. 1. Pseudocode of Algorithm Learn^{++}

subsets are chosen. The distribution itself is obtained by normalizing a set of weights assigned to each instance based on the classification performance of the classifier on that instance. In general, instances that have not yet been learned or seen are given higher weights to increase their chance of being selected into the next training data.

At each iteration t, the distribution $\boldsymbol{D_t}$ is obtained by normalizing the weights $\boldsymbol{w_t}$ of the instances updated based on their classification by the previous ensemble (step 1 of the inner **Do** loop). A new training subset $\boldsymbol{TR_t}$ is drawn according to $\boldsymbol{D_t}$ and the **Base-Classifier** is trained with $\boldsymbol{TR_t}$ to generate the hypothesis h_t (step 2). The error ε_t of this hypothesis is calculated on the current training data $\boldsymbol{S_k}$ by adding the distribution weights of the misclassified instances (step 3). If $\varepsilon_t>1/2$, current h_t is deemed too weak, and is replaced with a new h_t, generated from a fresh $\boldsymbol{TR_t}$. If $\varepsilon_t<1/2$, the current hypothesis is retained, and all hypotheses generated during the previous t iterations are combined, using an appropriate combination schemes described later, to construct the *composite hypothesis* H_t (step 4). The composite error E_t made by H_t is determined by adding the distribution weights of all instances misclassified by the ensemble (step 5). The normalized composite error, B_t is computed, and used in updating the weights $w_t(i)$, which are then used in computing the next distribution D_{t+1}, which in turn is used in selecting the next training subset $\boldsymbol{TR_{t+1}}$. Once T_k hypotheses are generated for each database $\boldsymbol{DB}_k$, the final hypothesis H_{final} is obtained by combining all hypotheses by using one of the combination rules described below.

While Learn^{++} uses similar ensemble generation structure as AdaBoost, there are several key differences: AdaBoost runs on a single database; it has no distribution re-initialization; and it stops and aborts if $\varepsilon_t > ½$ for any h_t. Most importantly, AdaBoost is designed to improve the performance of a weak classifier, for which it uses the performance of the current single hypothesis h_t to update its weight distribution [4]. Learn^{++}, however, creates a composite hypothesis H_t representing the ensemble decision, and uses the *ensemble* performance to update its weight distribution. This allows a more efficient incremental learning ability, particularly if the new database introduces instances from a previously unseen class. When instances of a new class are introduced, an existing ensemble H_t – not yet seen instances of the new class, is bound to misclassify them, forcing the algorithm to focus on these instances that carry novel information. For a weight update rule based on the performance of h_t only, the training performance of the first h_t on instances from the new class is independent of the previously generated classifiers. Therefore, the new h_t is not any more likely to misclassify new class instances, which then causes AdaBoost to focus on other *difficult to learn* instances, such as outliers, rather than the instances with novel information content. It is this claim that we investigate in this effort.

Learn^{++} was previously shown to be capable of incremental learning, however, it's incremental learning ability has not been compared to that of AdaBoost. Given that AdaBoost was not originally designed for incremental learning, one can argue whether it is fair to compare AdaBoost to an algorithm that is designed for incremental learning. However, Learn^{++} shares much of its algorithmic detail with AdaBoost. The main difference is the distribution update rule being based on ensemble decision, rather than the previous hypothesis. Therefore, a questions of particular interest is as follows: is the incremental learning ability of Learn++ primarily due to creating and combining an ensemble of classifiers, or is it due to the strategic selection of the distribution update rule? If incremental learning ability is provided primarily by combining an ensemble of classifiers, then AdaBoost should also be able to learn incrementally.

In order to answer this question, and establish the true impact of the difference in distribution update rules, the two algorithms must be made equivalent in all other aspects. Therefore, we slightly modify AdaBoost to allow it to generate additional ensembles with new data, using the same distribution re-initialization as Learn^{++} (but retaining its own single-hypothesis-based distribution update rule). We also allow AdaBoost to generate a replacement hypothesis for any h_t that does not satisfy $\varepsilon_t < ½$ requirement. Therefore, the only difference left between the modified AdaBoost and Learn^{++} is the distribution update rule.

3 Combination Rules

Properties of different combination rules for ensemble systems have been well researched [7,8]. While the best combination rule is often application dependent, certain rules, such as the sum, weighted majority, and decision templates have repeatedly shown superior performance over others, and hence are used more often. Both AdaBoost and Learn^{++} were originally designed to be used with weighted majority voting. However, in this study, we also compare each with different combination rules, to determine whether the combination rule (in addition to distribution update rule) has any effect on incremental learning performance.

Some combination rules, namely, simple and weighted majority voting (VMW), only need access to class labels. Others need the degree of support given by the classifier to each class. For the first group, let us define the decision of the t^{th} classifier as the binary valued $d_{t,j} \in \{0,1\}$, $t=1,\ldots,T$ and $j=1,\ldots,C$, where T is the number of classifiers and C is the number of classes. If classifier h_t correctly identifies class ω_j, $d_{t,j}=1$, and zero otherwise. For other rules, we have continuous valued $d_{t,j} \in [0,1]$, which represent the degree of support given by classifier h_t to class ω_j. For any given classifier, these supports are normalized to add up to 1 over different classes, and are often interpreted as class conditional posterior probabilities, $P(\omega_j|\mathbf{x})$.

We use the *decision profile matrix* [9], to formalize all combination rules: for an instance $\mathbf{x}$, the decision profile matrix $DP(\mathbf{x})$, consists of the elements $d_{t,j}$. The t^{th} row of $DP(\mathbf{x})$ is the support given by the t^{th} classifier to each class, and the j^{th} column is the support received by class ω_j from all classifiers. The total support for each class is obtained as a simple function of the supports received by individual classifiers. We represent the total support received by class ω_j as

$$\mu_j(x) = \Im[d_{1,j}(x),\cdots,d_{T,j}(x)]\cdot \tag{7}$$

where $\Im$ (.) is the combination function. We discuss the sum, product, median, simple majority, weighted majority, and decision template combination rules.

In an ensemble system, the final decision is the class that receives the largest support from the ensemble. Let ω_k be the winning class. In simple majority voting, ω_k is the class that is selected by most number of classifiers. For binary valued $d_{t,j}$,

$$\sum_{t=1}^{T} d_{i,k} = \max_{j=1}^{c} \sum_{t=1}^{T} d_{t,j} \tag{8}$$

If some classifiers are known to be more competent than others, giving higher weights to those classifiers may improve the performance. Denoting the voting weight for classifier h_t with V_t, weighted majority voting (WMV) can be obtained as

$$\sum_{t=1}^{T} V_t d_{t,k} = \max_{j=1}^{c} \sum_{t=1}^{T} V_t d_{t,j} \cdot \tag{9}$$

In original Learn^{++} and AdaBoost, these weights are inversely proportional to the training errors of h_t:

$$V_t = \log\left((1-\varepsilon_t)/\varepsilon_t\right) \tag{10}$$

The sum, product and median rules are similar, defined by the following expressions, respectively

$$\mu_j(x) = \frac{1}{T}\sum_{t=1}^{T} d_{t,j}(x) \tag{11}$$

$$\mu_j(x) = \frac{1}{T}\prod_{t=1}^{T} d_{t,j}(x) \tag{12}$$

$$\mu_j(x) = \underset{t=1\ldots T}{median}\{d_{t,j}(x)\} \tag{13}$$

In each case, the ensemble decision is the class ω_k for which the total support $\mu_j(\mathbf{x})$ is highest.

Perhaps the most sophisticated combination rule that uses all supports given by all classifiers to all classes is Kuncheva's decision templates [9]. For each class ω_j, the decision template DT_j is the average of all decision profiles in training data $\boldsymbol{X_j}$

$$DT_j = \frac{1}{M_j}\sum_{\mathbf{x}\in X_j} DP(\mathbf{x}) \tag{14}$$

where $\boldsymbol{X_j}$ is the set of instances coming from class ω_j; and M_j is the cardinality of $\boldsymbol{X_j}$. The class ω_k whose decision template is closest to the decision profile of the current instance, e.g., using squared *Euclidean* distance, is chosen as the ensemble decision. The closest match then decides on the label of **x.**

$$\mu_j(x) = 1 - \frac{1}{T\times C}\sum_{t=1}^{T}\sum_{k=1}^{C}\left[DT_j(t,k) - d_{t,k}(x)\right]^2 \tag{15}$$

4 Simulation Results

We present and compare simulation results of Learn^{++} and AdaBoost.M1 on several real-world and benchmark databases, using six different combination rules on each. All results are given with 95% confidence interval, obtained through 10 fold cross validation. Each database was partitioned into n sets: $\boldsymbol{S}_1$~$\boldsymbol{S}_n$ for training, where each set introduced one or more new classes, and an additional ***TEST*** set for validation, which included instances from all classes. Ensemble generalization performances after each training session $\boldsymbol{TS_1}$ ~ $\boldsymbol{TS_n}$ (trained on $\boldsymbol{S}_1$~$\boldsymbol{S}_n$ separately, and tested on ***TEST***) are presented below. Multilayer perceptrons were used as base classifiers.

4.1 Ultrasonic Weld Inspection (UWI) Dataset

The UWI dataset was obtained from ultrasonic inspection of submarine hull welding regions. The welding regions, also known as heat-affected zones, are highly susceptible to growing a variety of defects, including potentially dangerous cracks. The discontinuities within the material, such as air gaps due to cracks, cause the ultrasonic wave to be reflected back, and received by the transducer. The reflected ultrasonic wave, also called an A-scan, serves as the signature pattern of the discontinuity, which is then analyzed to determine whether it was originated from a crack. However, this analysis is hampered by the presence of other types of discontinuities, such as porosity, slag and lack of fusion (LOF), all of which generate very similar A-scans, resulting in a very challenging database with highly overlapping classes. The data distribution and percent generalization performances of both algorithms are shown in Tables 1 and 2 respectively. The number of classifiers used to train datasets S_1, S_2 and S_3 were set as 3, 5 and 7, respectively, and kept constant for all experiments. The best performance for each algorithm at the end of TS_3 is shown in bold.

Table 1. Data distribution for UWI dataset

Dataset ↓	*Crack*	*Slag*	*LOF*	*Porosity*
S_1	300	300	0	0
S_2	200	200	200	0
S_3	150	150	137	99
TEST	200	200	150	125

Table 2. Percent test performance of AdaBoost.M1 and Learn^{++} on UWI dataset

	AdaBoost.M1			**Learn^{++}**		
	TS_1	TS_2	TS_3	TS_1	TS_2	TS_3
SUM	49.6±1.2	59.1±1.2	57.8±1.6	52.0±0.6	65.7±0.8	70.5±0.8
PRODUCT	48.8±1.2	59.0±1.1	57.1±2.6	51.4±0.6	62.4±0.6	65.0±0.7
MEDIAN	49.1±1.3	58.7±1.2	56.6±1.6	51.5±0.7	65.1±0.8	**70.8±0.6**
M. VOTING	49.0±0.8	58.8±1.1	58.5±1.1	51.4±0.6	65.5±0.7	70.3±0.8
WMV	49.6±1.6	59.2±1.1	**59.3±1.0**	51.4±0.9	65.1±1.0	70.6±0.5
DT	49.0±0.8	59.6±1.0	58.5±1.1	52.1±0.8	65.2±0.6	68.8±0.6

Table 2 shows that both algorithms were able to learn incrementally from the new data, as indicated by the improved generalization performances from one training session to the next. Learn^{++} outperformed AdaBoost, however, with statistical significance, on all combination rules. Furthermore, the performance of AdaBoost mildly deteriorated in TS_3, compared to its performance on the previous session, TS_2. AdaBoost could not learn the new class information, at least using the number of classifiers specified. As mentioned earlier, this is attributed to the composite hypothesis based weight update rule of Learn^{++}. The performance differences among different combination rules were mostly statistically insignificant for both algorithms.

4.2 Volatile Organic Compound (VOC) Dataset

This database was generated from responses of six quartz crystal microbalances (QCMs) to various concentrations of five volatile organic compounds, including ethanol (ET), xylene (XL), octane (OC), toluene (TL), and trichloroethylene (TCE). The data distribution, indicating a new class introduced with each dataset, is shown in Table 3, and the mean generalization performances of AdaBoost.M1 and Learn^{++} for the VOC database are presented in Table 4. $\boldsymbol{S_1}$ had instances from ET, OC and TL, $\boldsymbol{S_2}$ added instances primarily from the new class TCE (and fewer instances from the previously three), and $\boldsymbol{S_3}$ added instances from XL (and fewer instances from the previous four). The number of classifiers used to train datasets $\boldsymbol{S_1}$, $\boldsymbol{S_2}$ and $\boldsymbol{S_3}$ were chosen as 2, 3 and 5, respectively, and kept constant for all experiments.

Table 3. Data distribution for VOC dataset

Dataset ↓	*ET*	*OC*	*TL*	*TCE*	*XL*
S_1	20	20	40	0	0
S_2	10	10	10	25	0
S_3	10	10	10	15	40
TEST	24	24	52	24	40

Table 4. Percent test performance of AdaBoost.M1 and Learn^{++} on VOC dataset

	AdaBoost.M1			**Learn^{++}**		
	TS_1	TS_2	TS_3	TS_1	TS_2	TS_3
SUM	61.1±0.7	63.8±2.0	67. ±7.5	62.0±1.3	71.4±0.5	83.2±3.4
PRODUCT	60.3+1.4	61.2±5.3	**70.9±3.7**	60.8±0.5	62.4±3.2	71.4±4.0
MEDIAN	60.6±0.8	59.3±6.5	67.2±4.5	61.2±0.4	66.2±1.1	79.8±3.3
M.VOTING	58.9±3.4	63.9±1.8	67.7±4.5	61.4±0.4	69.9±1.3	**85.1±1.1**
WMV	60.3±1.1	59.2±6.8	69.9±2.5	61.5±0.4	71.6±0.6	**85.2±1.5**
DT	60.2±1.1	62.6±2.8	63.3±6.1	61.2±0.5	67.2±1.2	76.7± 2.4

While both algorithms achieved incremental learning, Learn^{++} performed significantly better than AdaBoost.M1 on all combination rules, and usually with smaller confidence intervals. As expected, majority voting, weighted majority voting and the sum rule in general performed better than others.

4.3 Wine

The wine database from the UCI repository [10] is commonly used as a benchmark dataset. The dataset describes chemical analysis of 13 constituents found in three types of Italian wines, derived from three different cultivars of the same region. The data distribution and the test performances of both algorithms are shown in Tables 5 and 6 respectively. $\boldsymbol{S_1}$ had instances only from classes 1 and 2, whereas $\boldsymbol{S_2}$ introduced class 3. The number of classifiers used to train datasets $\boldsymbol{S_1}$ and $\boldsymbol{S_2}$ were set as 2 and 4, respectively, and kept constant for all experiments.

Table 5. Data distribution for Wine dataset

Dataset ↓	*Wine$_1$*	*Wine$_2$*	*Wine$_3$*
S$_1$	30	40	0
S$_2$	10	10	30
TEST	71	28	21

Table 6. Percent test performance of AdaBoost.M1 and Learn^{++} on Wine dataset

	AdaBoost.M1		**Learn^{++}**	
	TS$_1$	*TS$_2$*	*TS$_3$*	*TS$_1$*
SUM	60.2±6.1	77.8±9.7	61.2±4.1	82.2±6.1
PRODUCT	59.1±8.7	81.4±10.2	60.5±2.6	82.1±6.2
MEDIAN	59.0±7.2	68.1±16.8	64.5±2.4	84.0±9.1
M.VOTING	58.8±6.6	77.1±14.3	60.0±2.4	82.9±8.5
WMV	54.7±8.3	76.0±12.9	62.8±3.1	82.6±6.6
DT	62.1±3.8	73.4±16.8	60.7 ±3.4	70.7±4.8

As in previous datasets, both algorithms were able to learn incrementally from the new data, as seen by the improved generalization performances from one training session to the next. Learn^{++}, however, performed significantly better than AdaBoost.M1 on all combination rules except the decision templates, and usually with smaller (though still somewhat large) confidence intervals. We should add however that due to somewhat large confidence intervals, the performance differences among different combination rules were not statistically significant.

4.4 Optical Character Recognition Database

Also obtained from the UCI repository [10], OCR database consists of handwritten numeric characters, 0 through 9, digitized on an 8x8 grid creating 64 attributes for 10 classes. The data distribution and the mean performances of the two algorithms are shown in Tables 7 and 8, respectively. Note that the data distribution was made deliberately challenging, specifically designed to test the algorithms' ability to learn multiple new classes with each dataset, while retaining the knowledge of previously learned classes. In particular, ***S_1*** consisted of classes 0,1,2,5,6 and 9, S_2 added classes 3 and 4, and ***S_3*** introduced classes 7 and 8, but removes instances from classes 0 and 1. The number of classifiers used to train datasets ***S_1, S_2*** and ***S_3*** were set as 3, 3 and 3, respectively, and kept constant for all experiments. Interesting observations can be made from the generalization performances of AdaBoost and Learn^{++}.

Table 7. Data distribution for OCR dataset

Dataset ↓	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
S$_1$	250	250	250	0	0	250	250	0	0	250
S$_2$	100	100	100	250	250	100	100	0	0	100
S$_3$	0	0	50	150	150	50	50	400	400	0
TEST	100	100	100	100	100	100	100	100	100	100

Table 8. Percent test performance of AdaBoost.M1 and Learn^{++} on OCR dataset

	AdaBoost.M1			**Learn^{++}**		
	TS$_1$	***TS$_2$***	***TS$_3$***	***TS$_1$***	***TS$_2$***	***TS$_3$***
SUM	57.4±2.0	71.1±2.6	66.2±2.2	59.2±0.2	77.0±1.2	88.5±1.5
PRODUCT	58.0±0.5	70.3±2.6	64.5±2.8	59.8±1.1	78.2±0.5	88.4±1.4
MEDIAN	55.3±4.4	70.5±2.7	**67.0±1.9**	59.0±0.2	77.9±0.2	**90.5±1.8**
M.VOTING	58.1±0.5	71.9±2.7	65.0±2.5	58.6±0.3	75.5±1.0	82.0±0.7
WMV	57.4±2.0	68.8±1.8	**67.0±3.8**	59.2±0.1	77.2±0.4	89.3±1.0
DT	58.3±0.5	72.5±1.4	65.7±3.4	59.3±0.4	79.8±0.5	82.5±1.7

Similar to the previous databases discussed above, Learn^{++} outperformed AdaBoost, with statistical significance, on all combination rules. It is interesting to observe that AdaBoost incrementally learns the second database, however it displays substantial amount of forgetting from second to third training session. This indicates that AdaBoost is having difficulty in learning new classes, and at the same time retaining the information it had previously learned, particularly if subsequently generated classifiers are no longer trained with instances of previously seen classes. Conversely, Learn^{++} was able to achieve upper 80% to lower 90% classification performance, using any of the sum, product, median and weighted majority voting combination rules. Considering that the database was designed to test the incremental learning and knowledge retaining ability of the algorithm (by leaving instances of certain classes out), we can conclude that Learn^{++} places itself more favorably along the plasticity–stability spectrum.

5 Conclusion and Discussion

Our results indicate that AdaBoost.M1 can indeed learn incrementally from new data; however, its effectiveness is limited by its single-hypothesis-based distribution update rule. We should quickly point out that this is not a short coming of AdaBoost, as the algorithm was not originally intended for incremental learning, but rather to allow weak classifiers learn in an ensemble structure. As consistently seen in all results, and in particular in hostile learning environments, where the consecutive databases may introduce instances of new classes and/or remove instances from previously seen classes, the ensemble-based distribution update rule of Learn^{++} provides substantial performance improvement.. Therefore, we conclude that the ensemble based distribution update rule is indeed crucial in achieving efficient incremental learning.

We also note that Learn^{++} achieved narrower confidence intervals in its performances. This is significant, because a narrower confidence interval indicates better stability and robustness, qualities of considerable concern in incremental learning. Improved generalization performance along with a narrower confidence interval shows that Learn^{++} can achieve a delicate balance on the stability-plasticity spectrum.

We should note that despite its relative inferior performance in incremental learning, AdaBoost is still a strong contender: it has certainly shown promise in incremental learning of certain applications, including learning new classes. We believe that AdaBoost can still be used for incremental learning applications where the learning environment is less hostile than the one we created in our simulations. Also, since we were interested in efficient incremental learning, the ensemble sizes were kept to minimum. If AdaBoost were allowed to generate additional classifiers, it could have achieved better performances. The incremental learning ability of AdaBoost under such cases is currently being investigated.

Unlike the distribution update rule, the choice of specific combination rule does not appear to be very influential in the incremental learning performance of either algorithm. While there were some differences – sometimes significant – such differences were not consistent, and we believe that the impact of a particular combination rule is relatively minor, compared to that of the distribution update rule.

References

1. R. French, “Catastrophic forgetting in connectionist networks,” *Trends in Cognitive Sciences*, vol. 3, no.4, pp. 128-135, 1999.
2. S.Grossberg, “Nonlinear neural networks: principles, mechanisms and architectures,” Neural Networks, vol. 1, no. 1, pp. 17-61, 1988.
3. G.A. Carpenter, S. Grossberg, N. Markuzon, J.H. Reynolds, and D.B. Rosen, “Fuzzy ARTMAP: A neural network architecture for incremental learning of analog multidimensional maps,” IEEE Trans. on Neural Networks, vol. 3, no. 5, pp.698-713, 1992.
4. Y. Freund, R. Schapire, Decision-theoretic generalization of on-line learning and an application to boosting, *J. Comp. Sys. Sci.*, vol. 55, no. 1, pp. 119-13, 1997.
5. R. Polikar, L. Udpa, S. Udpa, V. Honavar., “Learn^{++}: An incremental learning algorithm for supervised neural networks,” *IEEE Trans. on System, Man and Cybernetics (C),* vol. 31, no. 4, pp. 497-508, 2001.
6. R. Polikar, J. Byorick, S. Krause, A. Marino, M. Moreton, “Learn++: A classifier independent incremental learning algorithm for supervised neural networks,” Proc. of Int. Joint Conference on Neural Networks (IJCNN 2002), vol. 2, pp. 1742-1747, Honolulu, HI, 12-17 May 2002.
7. J. Kittler, M. Hatef, R. P.W. Duin, and J. Matas, “On combining classifiers,” IEEE Trans. on Pattern Analy. and Machine Int., vol. 20, no. 3, pp.226-239, 1998.
8. L.I. Kuncheva. Combining Pattern Classifiers: Methods and Algorithms. John Wiley & Sons, N.J., 2004.
9. L. I. Kuncheva, J. C. Bezdek, and R. P. W. Duin, "Decision templates for multiple classifier fusion: an experimental comparison," Pattern Rec., vol. 34, no. 2, pp. 299-314, 2001.
10. C.L. Blake and C.J. Merz, Univ. of California, Irvine, Repository of Machine Learning Databases at Irvine, CA.