Learn++.MT: A New Approach to Incremental Learning

Michael Muhlbaier, Apostolos Topalis, and Robi Polikar

Rowan University, Electrical and Computer Engineering Department
201 Mullica Hill Rd., Glassboro, NJ 08028, USA
{muhl1565,topa4536}@students.rowan.edu, polikar@rowan.edu

Abstract. An ensemble of classifiers based algorithm, Learn++, was recently introduced that is capable of incrementally learning new information from datasets that consecutively become available, even if the new data introduce additional classes that were not formerly seen. The algorithm does not require access to previously used datasets, yet it is capable of largely retaining the previously acquired knowledge. However, Learn++ suffers from the inherent "out-voting" problem when asked to learn new classes, which causes it to generate an unnecessarily large number of classifiers. This paper proposes a modified version of this algorithm, called Learn++.MT that not only reduces the number of classifiers generated, but also provides performance improvements. The out-voting problem, the new algorithm and its promising results on two benchmark datasets as well as on one real world application are presented.

1 Introduction

It is well known that the amount of training data available and how well the data represent the underlying distribution are of paramount importance for an automated classifier's satisfactory performance. For many applications of practical interest, obtaining such adequate and representative data is often expensive, tedious, and time consuming. Consequently, it is not uncommon for the entire data to be obtained in installments, over a period of time. Such scenarios require a classifier to be trained and incrementally updated – as new data become available – where the classifier needs to learn the novel information provided by the new data without forgetting the knowledge previously acquired from the data seen earlier. This raises the so-called stability-plasticity dilemma [1]: a completely stable classifier can retain knowledge, but cannot learn new information, whereas a completely plastic classifier can instantly learn new information, but cannot retain previous knowledge. Many popular classifiers, such as the ubiquitous multilayer perceptron (MLP) or the radial basis function networks, are not structurally suitable for incremental learning, since they are "completely stable" classifiers. The approach generally followed for learning from new data involves discarding the existing classifier, combining the old and the new data and training a new classifier from scratch using the aggregate data. This causes the previously learned information to be lost, a phenomenon known as catastrophic forgetting [2]. Furthermore, training with the combined data may not even be feasible, if the previously used data are lost, corrupted, prohibitively large, or otherwise unavailable.

F. Roli, J. Kittler, and T. Windeatt (Eds.): MCS 2004, LNCS 3077, pp. 52–61, 2004.

We have recently introduced an algorithm, called Learn++, capable of learning incrementally, even under hostile learning conditions: not only does Learn++ assume the previous data to be no longer available, but it also allows additional classes to be introduced with new data, while retaining the previously acquired knowledge.

Learn++ is an ensemble approach, inspired primarily by the AdaBoost algorithm. Similar to AdaBoost, Learn++ also creates an ensemble of (weak) classifiers, each trained on a subset of the current training dataset, and later combined through weighted majority voting. Training instances for each classifier are drawn from an iteratively updated distribution. The main difference is that the distribution update rule in AdaBoost is based on the performance of the previous hypothesis [3], which focuses the algorithm on difficult instances, whereas that of Learn++ is based on the performance of the entire ensemble [4], which focuses this algorithm on instances that carry novel information. This distinction gives Learn++ the ability to learn new data, even when previously unseen classes are introduced. As new data arrive, Learn++ generates additional classifiers, until the ensemble learns the novel information. Since no classifier is discarded, previously acquired knowledge is retained. Other approaches suggested for incremental learning, a bibliography of ensemble systems and their applications can be found in and within the references of [4 ~9].

As reported in [4,5], Learn++ works rather well on a variety of real world problems, though there is much room for improvement. An issue of concern is the relatively large number of classifiers required for learning instances coming from a new class. This is because, when a new dataset introduces a previously unseen class, new classifiers are trained to learn the new class; however, the existing classifiers continue to misclassify instances from the new class. Therefore, the decisions of latter classifiers that recognize the new class are out-voted by the previous classifiers that do not recognize the new class, until a sufficient number of new classifiers are generated that recognize the new class. This leads to classifier proliferation.

In this contribution, we first describe the out-voting problem associated with the original Learn++, propose a modified version of the algorithm to address this issue, and present some preliminary simulation results on three benchmark datasets.

2 Learn++.MT

In ensemble approaches that use a voting mechanism for combining classifier outputs, each classifier votes on the class it predicts [10, 11]. The final classification is then determined as the class that receives the highest total vote from all classifiers. Learn++ uses weighted majority voting [12], where each classifier receives a voting weight based on its training performance. This works well in practice for most applications. However, for incremental learning problems that involve introduction of new classes, the voting scheme proves to be unfair towards the newly introduced class: since none of the previously generated classifiers can pick the new class, a relatively large number of new classifiers that recognize the new class are needed, so that their total weight can out-vote the first batch of classifiers on instances of the new class. This in return populates the ensemble with an unnecessarily large number of classifiers. Learn++.MT is specifically designed to address the classifier proliferation issue. The novelty in Learn++.MT is the way by which the voting weights are determined. Learn++.MT also uses a set of voting weights based on the classifiers' performances,

however, these weights are then adjusted based on the classification of the specific instance at the time of testing, through dynamic weight voting (DWV).

For any given test instance, Learn++.MT compares the class predictions of each classifier and cross-references them against the classes on which they were trained. If a subsequent ensemble overwhelmingly chooses a class it has seen before, then the voting weights of those classifiers not trained with that class are proportionally reduced. As an example, assume that an ensemble has seen classes 1 and 2, and a second ensemble has seen classes 1, 2 and 3. For a given instance, if the second ensemble (trained on class 3) picks class 3, the classifiers in the first ensemble (which has not seen class 3) reduce their voting weights in proportion to the confidence of the second ensemble. In other words, when the algorithm detects that the new classifiers overwhelmingly choose a new class on which they were trained, the weights of the other classifiers which have not seen this new class are reduced. The Learn++.MT algorithm is given in Figures 1 and 2, and explained in detail below.

For each dataset ($\mathcal{D}_k$) that becomes available to Learn++.MT, the inputs to the algorithm are (i) a sequence of m_k training data instances x_i and their correct labels y_i, (ii) a classification algorithm **BaseClassifier**, and (iii) an integer T_k specifying the maximum number of classifiers to be generated using that database. If the algorithm is seeing its first database (k=1), a data distribution (D_t) – from which training instances will be drawn – is initialized to be uniform, making the probability of any instance being selected equal. If k>1 then the distribution is updated from the previous step based on the performance of the existing ensemble on the new data. The algorithm then adds T_k classifiers to the ensemble starting at $t=eT_k+1$ where eT_k denotes the number of classifiers that currently exist in the ensemble.

For each iteration t, the instance weights, w_t, from the previous iteration are first normalized (step 1) to create a weight distribution D_t. A hypothesis, h_t, is generated from a subset of $\mathcal{D}_k$ that is drawn from D_t (step 2). The error, ε_t, of h_t is then calculated; if $\varepsilon_t > ½$, the algorithm deems the current classifier, h_t, to be too weak, discards it, and returns to step 2, otherwise, calculates the normalized error β_t (step 3). The class labels of the training instances used to generate this hypothesis are then stored as CTr_t (step 4). The dynamic weight voting (DWV) algorithm is called to obtain the composite hypothesis, H_t, of the ensemble (step 5). H_t represents the ensemble decision of the first t hypotheses generated thus far. The error of the composite hypothesis, E_t is then computed and normalized (step 6). The instance weights w_t are finally updated according to the performance of H_t (step7) such that the weights of instances correctly classified by H_t are reduced (and those that are misclassified are effectively increased). This ensures that the ensemble focus on those regions of the feature space that are not yet learned, paving the way for incremental learning.

The inputs to the dynamic weight voting algorithm are (i) the current training data (during training) or any test instance, (ii) classifiers h_t, (iii) β_t, normalized error for each h_t, and (iv) the vector CTr_t containing the classes on which h_t has been trained. Classifier weights are first initialized (step 1), where each classifier receives a standard weight that is inversely proportional to its normalized error β_t so that those classifiers that performed well on their training data are given higher voting weights. A normalization factor Z_c is then created as the sum of the weights of all classifiers trained on instances from class c (step 2).

Input: For each dataset $\mathcal{D}_k$ $k=1,2,...,K$

- Sequence of i=1,…,m_k instances x_i with labels $y_i \in Y_k = \{1,...,c\}$
- Weak learning algorithm **BaseClassifier**.
- Integer T_k, specifying the number of iterations.

Do for $k=1,2,...,K$

If k=1 **Initialize** $w_1 = D_1(i) = 1/m$, $eT_1 = 0$ for all i.

Else Go to Step 5 to evaluate the current ensemble on new dataset $\mathcal{D}_k$, update weights, and recall current number of classifiers → $eT_k = \sum_{j=1}^{k-1} T_j$

Do for $t=eT_k+1,\ eT_k+2,\ldots,\ eT_k+T_k$:

1. Set $D_t = w_t \Big/ \sum_{i=1}^{m} w_t(i)$ so that D_t is a distribution.
2. **Call BaseClassifier** with a subset of $\mathcal{D}_k$ randomly chosen using D_t.
3. Obtain h_t : X → Y, and calculate its error: $\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$

   If $\varepsilon_t > ½$, discard h_t and go to step 2. Otherwise, compute normalized error as $\beta_t = \varepsilon_t/(1-\varepsilon_t)$.
4. $CTr_t = Y_k$, to save labels of classes used in training h_t.
5. **Call DWV** to obtain the composite hypothesis H_t.
6. Compute the error of the composite hypothesis $E_t = \sum_{i:H_t(x_i)\neq y_i} D_t(i)$
7. Set $B_t=E_t/(1-E_t)$, and update the instance weights:

$$w_{t+1}(i) = w_t \times \begin{cases} B_t, & \text{if } H_t(x_i) = y_i \\ 1, & \text{otherwise} \end{cases}$$

Call DWV to obtain the final hypothesis, H_{final}.

Fig. 1. Learn++.MT Algorithm.

For each instance, a preliminary per-class confidence factor $0<P_c<1$ is generated (step 3). P_c is the sum of weights of all the classifiers that choose class c divided by the sum of the weights of all classifiers trained with class c (which is Z_c). In effect, this can be considered as the ensemble assigned confidence of the instance for belonging to each of the c classes. Then, again for each class, the weights are adjusted for classifiers that have not been trained with that class, that is, the weights are lowered proportional to the ensemble's preliminary confidence on that class (step 4). The final / composite hypothesis is then calculated as the maximum sum of the weights that chose a particular class (step 5).

Input:

- Sequence of i=1,…, n training instances or any test instance x_i
- Classifiers h_t.
- Corresponding error values, β_t.
- Classes, CTr_t used in training h_t.

For $t=1,2,\ldots,T$ where T is the total number classifiers

1. **Initialize** classifier weights $W_t = \log(1/\beta_t)$
2. **Create** normalization factor, Z, for each class
$$Z_c = \sum_{t:c\in CTr_t} W_t, \text{ for } c=1,2,\ldots,C \text{ classes}$$
3. **Obtain** preliminary decision $P_c = \sum_{t:h_t(x_i)=c} W_t \Big/ Z_c$, for $c=1,2,\ldots,C$
4. **Update** voting weights $W_{t:c\notin CTr_t} = W_{t:c\notin CTr_t}(1-P_c)$, for $c=1,2,\ldots,C$
5. **Compute** final / composite hypothesis
$$H_{final}(x_i) = \arg\max_c \sum_{t:h_t(x_i)=c} W_t$$

Fig. 2. Dynamic Weight Voting Algorithm for Learn++.MT.

3 Learn++.MT Simulation Results

Learn++.MT has been tested on several databases. For brevity, we present results on two benchmark databases and one real-world application. The benchmark databases are the Wine database and the Optical Character Recognition database from UCI [13], and the real world application is a gas identification problem for determining one of five volatile organic compounds based on chemical sensor data. MLPs – normally incapable of incremental learning – were used as base classifiers on all three cases. Base classifiers were all single layer MLPs with 20~50 nodes and a rather generous error goal of 0.1 ~ 0.01 to ensure weak classifiers with respect to the difficulty of the underlying problem.

3.1 Wine Recognition Database

The Wine Recognition database features 3 classes with 13 attributes. The database was split into two training, a validation, and a test dataset. The data distribution is given in Table 1. In order to test the algorithms' ability to incrementally learn a new class, instances from class 3 are only included in the second dataset. Each algorithm was allowed to create a set number of classifiers (30) on each dataset. The optimal number of classifiers to retain for each dataset was automatically determined based on the maximum performance on the validation data. This process was applied 30 times on Learn++ and Learn++.MT to compare their generalization performance on the test data, the mean results of which are shown in Tables 2 and 3. Each row shows class-

by-class generalization performance of the ensemble on the test data after being trained with dataset $\mathcal{D}_k$, k=1,2. The last two columns are the average overall generalization performance over 30 simulation trials (on the entire test data which includes instances from all three classes), and the standard deviation of the generalization performances. The number of classifiers in the ensemble after each training session is given in parentheses.

Table 1. Wine Recognition database distribution.

Class→	1	2	3
$\mathcal{D}_1$	26	31	0
$\mathcal{D}_2$	13	16	32
Valid.	7	8	5
Test	13	16	11

Table 2. Learn++ performance results on Wine Recognition database.

Class→	1	2	3	Gen.	Std.
$\mathcal{D}_1$ (6)	99%	96%	-	71%	5.8
$\mathcal{D}_2$ (26)	100%	94%	24%	**77%**	**12.1**

Table 3. Learn++.MT performance results on Wine recognition database.

Class→	1	2	3	Gen.	Std.
$\mathcal{D}_1$ (5)	96%	95%	-	70%	6.0
$\mathcal{D}_2$ (6)	99%	87%	90%	**92%**	**5.0**

Tables 2 and 3 show that Learn++.MT not only incrementally learns the new class, but also outperforms its predecessor by 15% using a significantly fewer number of classifiers. The poor performance of Learn++ in the new class (class 3) is explained below within the context of larger database simulations.

3.2 Optical Character Recognition Database

The optical character recognition (OCR) database features 10 classes (digits 0 ~ 9) with 64 attributes. The database was split into four to create three training and a test subset, whose distribution can be seen in Table 4. In this case, we wanted to evaluate the performance of each algorithm on a fixed number of classifiers (rather than determining the number of classifiers via a validation set) so that they can be compared on equal number of classifiers. Each algorithm was allowed to create five classifiers with the addition of each dataset (total of 15 classifiers in three training sessions). The data distribution was deliberately made rather challenging, specifically designed to test the algorithms' ability to learn multiple new classes at once with each additional dataset while retaining the knowledge of previously learned classes. In this incremental learning problem, instances from only six of the ten classes are present in each subsequent dataset resulting in a rather difficult problem. Results previously

obtained using Learn++ on this data using less challenging data distributions was in the order of lower to mid 90% range [4,5]. Results from this test are shown in Tables 5 and 6, which is formatted similar to the previous tables.

Table 4. OCR data distribution.

Class→	0	1	2	3	4	5	6	7	8	9
$\mathcal{D}_1$	250	250	250	0	0	250	250	250	0	0
$\mathcal{D}_2$	150	0	150	250	0	150	0	150	250	0
$\mathcal{D}_3$	0	150	0	150	400	0	150	0	150	400
Test	110	114	111	114	113	111	111	113	110	112

Table 5. Learn++ performance results on OCR database.

Class→	0	1	2	3	4	5	6	7	8	9	Gen.	Std.
$\mathcal{D}_1$	99%	98%	99%	-	-	96%	99%	100%	-	-	59%	0.6%
$\mathcal{D}_2$	98%	98%	99%	32%	-	96%	99%	100%	60%	-	68%	1.8%
$\mathcal{D}_3$	98%	96%	99%	94%	22%	96%	99%	100%	90%	13%	**81%**	**4.0%**

Table 6. Learn++.MT performance results on OCR database.

Class→	0	1	2	3	4	5	6	7	8	9	Gen.	Std.
$\mathcal{D}_1$	95%	98%	98%	-	-	95%	99%	100%	-	-	58%	0.8%
$\mathcal{D}_2$	96%	95%	99%	95%	-	95%	98%	100%	98%	-	69%	0.6%
$\mathcal{D}_3$	67%	95%	92%	98%	83%	63%	98%	100%	95%	96%	**89%**	**0.7%**

Interesting observations can be made from these tables. First, we note that Learn++ was able to learn the new classes, 3 and 8, only poorly after they were first introduced in $\mathcal{D}_2$ but able to learn them rather well, when further trained with these classes in $\mathcal{D}_3$. Similarly, it performs rather poorly on classes 4 and 9 after they are first introduced in $\mathcal{D}_3$, though it is reasonable to expect that it would do well on these classes with additional training. More importantly however, Learn++.MT was able to learn new classes quite well in its first attempt. Finally, recall that the generalization performance of the algorithm is computed on the entire test data which included instances from all classes. This is why the generalization performance is only around 60% after the first training session, since the algorithms have seen only six of the 10 classes in the test data. Both Learn++ and Learn++.MT exhibit an overall increase of generalization performance as new datasets are introduced – and hence the ability of incremental learning. Learn++.MT, however, is able to learn not only faster, but better than Learn++, as demonstrated by the significant jump in generalization performance (81% to 89%).

3.3 Volatile Organic Compound Recognition Database

The Volatile Organic Compound (VOC) database is a real world dataset that consist of 5 classes (toluene, xylene, hectane, octane and ketone) with 6 attributes coming

from six (quartz crystal microbalance type) chemical gas sensors. The dataset was divided into three training and a test dataset. The distribution of the data is given in Table 7, where a new class was introduced with each dataset.

Table 7. Volatile Organic Compounds database.

Class→	1	2	3	4	5
$\mathcal{D}_1$	20	0	20	0	40
$\mathcal{D}_2$	10	25	10	0	10
$\mathcal{D}_3$	10	15	10	40	10
Test	24	24	24	40	52

Again both algorithms were incrementally trained with three subsequent training datasets. In this experiment, both algorithms were allowed to generate as many classifiers as necessary to obtain their maximum performance. Learn++ generated a total of 36 classifiers to achieve its best performance. Learn++.MT, however, not only generated only 16 classifiers, but it also provided significant improvement in generalization performance.

Table 8. Learn++ performance results on VOC database.

Class	1	2	3	4	5	Gen	Std.
$\mathcal{D}_1$ (6)	90%	-	91%	-	88%	55%	1.3%
$\mathcal{D}_2$ (12)	89%	61%	95%	-	87%	64%	4.3%
$\mathcal{D}_3$ (18)	82%	95%	94%	12%	83%	**69%**	**6.6%**

Table 9. Learn++.MT performance results on VOC database.

Class	1	2	3	4	5	Gen	Std.
$\mathcal{D}_1$ (6)	90%	-	89%	-	89%	54%	2.0%
$\mathcal{D}_2$ (4)	86%	85%	93%	-	75%	63%	3.5%
$\mathcal{D}_3$ (6)	81%	96%	91%	83%	67%	**81%**	**2.3%**

The results averaged over 20 trials are given in Tables 8 and 9. We note that both algorithms provide a performance characteristic that is similar to those obtained with the previous databases. Specifically, Tables 8 and 9 show a significant increase from Learn++ to Learn++.MT on the average generalization performance. Furthermore, Learn++.MT was able to accomplish its performance using 20 fewer classifier, and learning each new class faster than its predecessor.

4 Conclusions and Discussions

In this paper we presented Learn++.MT, a modified version of our previously introduced incremental learning algorithm, Learn++. The novelty of the new algorithm is its use of preliminary confidence factors in assigning voting weights, based on a

cross-reference of the classes that have been seen by each classifier during training. Specifically, if a majority of the classifiers that have seen a class votes on that class, the voting weights of those classifiers who have not seen that class are reduced in proportion to the preliminary confidence. This allows the algorithm to dynamically adjust the voting weights for each test instance. The approach overcomes the out-voting problem inherent in the original version of Learn++ and prevents proliferation of unnecessary classifiers. The new algorithm also provided substantial improvements on the generalization performance on all datasets we have tried so far. We note that these improvements are more significant in those cases where one or several new classes are introduced with subsequent datasets.

It is also worth noting that, Learn++.MT is more robust than its predecessor. One of the reasons why Learn++ is having difficulty in learning a new class when first presented is due to difficulty in choosing the strength of the base classifiers. If we choose too weak classifiers, the algorithm is unable to learn. If we choose too strong classifiers, the training data are learned very well, resulting in very low β values which then causes very high voting weights, and hence even a more difficult out-voting problem. This explains why Learn++ requires larger number of classifiers or repeated training to learn the new classes. Learn++.MT, by significantly reducing the effect of the out-voting problem, improves the robustness of the algorithm, as the new algorithm is substantially more resistant to more drastic variations in the classifier architecture and parameters (error goal, number of hidden layer nodes, etc.).

We should also note however, while we have used MLPs as base classifiers, both algorithms are in fact independent of the type of the base classifier used, and can learn incrementally with any supervised classifier that lacks this ability. In fact, the classifier independence of Learn++ was demonstrated and reported in [5].

Further optimization of the distribution update rule, the selection of voting weights, as well as validation of the techniques on a broader spectrum of applications are currently underway.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. ECS-0239090, “CAREER: An Ensemble of Classifiers Approach for Incremental Learning.”

References

1. S. Grossberg, “Nonlinear neural networks: principles, mechanisms and architectures,” Neural Networks, vol. 1, no. 1, pp. 17-61, 1988.
2. R. French, “Catastrophic forgetting in connectionist networks,” Trends in Cognitive Sciences, vol. 3, no.4, pp. 128-135, 1999.
3. Y. Freund and R. Schapire, “A decision theoretic generalization of on-line learning and an application to boosting,” Computer and System Sci., vol. 57, no. 1, pp. 119-139, 1997.
4. R. Polikar, L. Udpa, S. Udpa, and V. Honavar, “Learn++: An incremental learning algorithm for supervised neural networks,” IEEE Trans. on Sys., Man and Cyber. (C), vol. 31, no. 4, pp. 497-508, 2001.

5. R. Polikar, J. Byorick, et al., "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, 2002.
6. M. Lewitt and R. Polikar, "An ensemble approach for data fusion with Learn++," 4th Int. Work. on Mult. Classifier Sys. LNCS (T. Windeatt and F. Roli, eds), vol. 2709, pp. 176-185, Springer: New York, NY, 2002.
7. J. Ghosh, "Multiclassifier systems: back to the future," 3rd Int. Work. on Mult. Classifier Sys., LNCS (J. Kitler & F. Roli, eds), vol. 2364, p. 1-15, Springer: New York, NY, 2002.
8. L.I. Kuncheva, "Switching between selection and fusion in combining classifiers: an experiment," IEEE Trans. on Sys., Man and Cyber., vol. 32(B), no. 2, pp. 146-156, 2002.
9. T. Windeatt and F. Roli (eds), In Proc. 4th Int. Workshop on Multiple Classifier Systems (MCS2003), LNCS, vol. 2709, Springer: New York, NY, 2002.
10. J. Kittler, M. Hatef, R.P. Duin, J. Matas, "On combining classifiers," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 20, no.3, pp. 226-239, 1998.
11. L.I. Kuncheva, "A theoretical study on six classifier fusion strategies, " IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 2, pp. 281-286, 2002.
12. N. Littlestone and M. Warmuth, "Weighted majority algorithm," Information and Computation, vol. 108, pp. 212-261, 1994.
13. C.L. Blake and C.J. Merz, UCI Repository of Machine Learning Databases at Irvine, CA: Available: http://www.ics.uci.edu/~mlearn/MLRepository.html