Proceedings of International Joint Conference on Neural Networks, Montreal, Canada, July 31 - August 4, 2005

Dynamically Weighted Majority Voting for Incremental Learning and Comparison of Three Boosting Based Approaches

Aliasgar Gangardiwala
Electrical and Computer Engineering
Rowan University
Glassboro, NJ 08028 USA
gangar34@students.rowan.edu

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

Abstract - We have previously introduced Learn++, an ensemble based incremental learning algorithm for acquiring new knowledge from data that later become available, even when such data introduce new classes. In this paper, we describe a modification to this algorithm, where the voting weights of the classifiers are updated dynamically based on the location of the test input in the feature space. The new algorithm provides improved performance, stronger immunity to catastrophic forgetting and finer balance to the stability-plasticity dilemma than its predecessor, particularly when new classes are introduced. The modified algorithm and its performance, as compared to Adaboost.M1 and the original Learn++, on real and benchmark datasets are presented.

I. Introduction

A. Incremental Learning

Supervised classifiers are effective and powerful learning tools for pattern recognition and machine learning applications. As most machine learning and pattern recognition professionals are painfully aware of, however, the generalization performance of any learning algorithm relies heavily on adequate and representative training data. Since data collection is an expensive, time consuming and a tedious process for most practical applications, such data are often acquired in small batches over time. Waiting for the entire dataset to be available for training may prove ineffective, uneconomical and inflexible. In such cases, it would be more desirable to train a classifier on available data and incrementally update the classifier as new data become available, without compromising the performance on previously learned data.

Learning new data incrementally without forgetting previously acquired knowledge raises the issue of stability-plasticity dilemma [1]: acquiring new knowledge requires plasticity, whereas retaining previously acquired knowledge requires stability. The challenge is then to achieve a meaningful balance between these two conflicting properties.

Many of the commonly used supervised classifiers such as multilayer perceptron (MLP), radial basis function, probabilistic neural networks, etc. are "very stable" classifiers, unable to learn new information. The practical approach generally taken with these classifiers for incremental learning is to discard the previously trained classifier, combine and use the entire training data accumulated thus far to create a new classifier from scratch. This approach effectively causes the previously learned information to be entirely lost, a phenomenon known as catastrophic forgetting [2,3].

For the purpose of this work, we define an incremental learning algorithm as one that has: (1) the capability of learning novel information content from consecutive datasets without requiring access to previously used data; (2) the capability of retaining previously learned knowledge; and (3) the ability to learn new classes introduced by new datasets.

Learn++, based on weighted majority voting of an ensemble of classifiers, satisfies the above listed criteria for incremental learning, yet resistant to aforementioned drawbacks [4, 5, 6]. In essence, the idea is to generate an ensemble of classifiers with the initial data, and generate additional classifiers as new datasets are acquired. We have recently noticed that the way in which the voting weights are assigned to classifiers based on their performance during training is suboptimal, because these weights are set during training and remain constant thereafter. A dynamic approach that assigns voting weights to classifiers based on the estimated performance of each classifier on that instance may be more optimal.

B. Ensemble of Classifiers and Weighted Majority Voting

Ensemble approaches have drawn much interest since Hansen and Salamon's seminal work [7]. In essence, a group of classifiers are trained using different distributions of training samples, and outputs of these classifiers are then combined in some manner to obtain the final classification rule.

Learn++ uses the synergistic power of such an ensemble for incremental learning of novel content provided by consecutive datasets. The algorithm was inspired from Freund and Schapire's adaptive boosting (Adaboost.M1) algorithm [8], which also was originally proposed for improving the performance of weak classifiers. It is based on the weighted majority voting [9] of hypotheses that are generated by sequentially training a set of weak classifiers on different distributions of the training data. Using weak classifiers allow different classifiers to make different errors, a combination of which through weighted majority voting then effectively averages out the individual errors resulting in a stronger classifier with a much improved generalization performance.

The original version of Learn++ followed the AdaBoost approach in determining voting weights, which were assigned during training depending on the classifiers' performance on their own training data. While this approach makes perfect sense when the entire data come from the same database, it does have a handicap when used in an incremental learning setting: since each classifier is trained to recognize (slightly) different portions of the feature space, classifiers performing well on a region represented by their training data may not do so well when classifying instances coming from different regions of the space. Therefore, assigning voting weights primarily on the training performance of each classifier is suboptimal. Estimating the potential performance of a classifier on a test instance using a statistical distance metric, and assigning voting weights based on these estimates may be more optimal. In this paper, we present a modified version of Learn++ along with its simulation results as compared to the original Learn++ to show the improvement on generalization performance and stability in incrementally learning new information content. Also new in this study, we evaluate AdaBoost for incremental learning and compare it both versions of Learn++. We show that, while not originally intended for such applications, AdaBoost is capable of incremental learning, albeit with a lower performance, efficiency and stability then either versions of Learn++.

Overviews of other ensemble and classifier combination techniques can be found in [10~15] and references within.

II. Learn++ with Dynamically Updated Voting Weights

Learn++, similar to AdaBoost, generates an ensemble of weak classifiers by choosing a subset of the training data from the database using an iteratively updated weight distribution rule, not to be confused with the voting weights. Learn++, however, combines all classifiers *at each iteration* to obtain a composite hypothesis before updating the distribution. The distribution update rule of Learn++ is then based on the performance of this composite hypothesis, which represents the performance of the entire ensemble that has been generated thus far. The distribution weights of those instances that are correctly identified by the ensemble are reduced. This distribution update rule is designed specifically to accommodate incremental learning of additional datasets, especially those that introduce previously unseen classes. The pseudocode of the algorithm is given in Fig.1.

For each database D_k, k= 1,…,K that becomes available, the inputs to Learn++ are: (1) labeled training data $S_k=\{[(\mathbf{x_i}, y_i)], i=1,\ldots,m_k\}$, where $\mathbf{x_i}$ is the training instance and y_i is the correct label; (2) a weak learning algorithm **BaseClassifier**; and (3) an integer T_k, the total number of weak classifiers to be generated. The BaseClassifier can be any supervised algorithm that can obtain a minimum of 50% classification performance on training data, ensuring the classifier is relatively weak, yet reasonably strong to have a meaningful performance. Using a weak classifier has the additional advantage of rapid learning, since the time-consuming fine-tuning step, which could potentially cause overfitting, is avoided.

Unless there is compelling reason to choose otherwise, the distribution weights are initialized to be uniform, so that all instances have the same probability of being selected into the first training subset. If $k>1$ (that is, new database has been introduced), a distribution initialization sequence re-initializes the data distribution (the If block in Fig. 1) based on the performance of the current ensemble on the new data.

At each iteration t, the distribution $\boldsymbol{D_t}$ is obtained by normalizing the weights $\boldsymbol{w_t}$ of the instances based on their individual classification by the current ensemble (step 1).

$$\boldsymbol{D_t} = \boldsymbol{w_t} \Big/ \sum_{i=1}^{m_k} w_t(i) \tag{1}$$

The training dataset S_k is divided into a training (TR_t) and a testing subset (TE_t) according to $\boldsymbol{D_t}$ (step 2). Learn++ than calls BaseClassifier (step 3) and trains it with TR_t to generate a weak hypothesis h_t. The error of this hypothesis is calculated on the current training data S_k by adding the distribution weights of the misclassified instances (step 4)

$$\varepsilon_t = \sum_{i:h_t(\mathbf{x})\neq y} D_t(i) = \sum_{i=1}^{m_k} D_t(i)\ [|\, h_t(\mathbf{x_i}) \neq y_i \,|] \tag{2}$$

where [|•|] is 1 if the predicate is true, and 0 otherwise. If $\varepsilon>1/2$, current h_t is deemed too weak, and is replaced with a new h_t generated from a fresh set of TR_t and TE_t. If $\varepsilon<1/2$, the current hypothesis is added to the previous hypotheses and all hypotheses generated during the previous t iterations are then combined using the weighted majority voting to construct the *composite hypothesis* H_t (step 5).

In original Learn++, the voting weights were calculated based on the error ε_t, so that hypotheses with lower error were given higher weights, resulting in classes predicted by these hypotheses to be weighted more heavily. Since the hypothesis weights are assigned prior to testing based on their individual training performance, this weight assignment is suboptimal. This is because, hypotheses are trained with different (and possibly overlapping) portions of the feature space, and it may not be reasonable to expect a classifier to perform well on test instances that may come from different portions of the feature space. This is not likely to be a major issue when only a single database is used (as in AdaBoost); however, it is a valid concern in an incremental learning setting. A more optimal rule would be to dynamically estimate which hypotheses are more likely to correctly classify any given instance and give them higher voting weights accordingly. Therefore, we modify the expression for composite hypotheses, representing ensemble decision, as

```
Input: For each dataset drawn from $D_k$ $k=1,2,\ldots,K$
  • Sequence of $m_k$ examples $S_k = \{(\mathbf{x}_i, y_i) \mid i = 1,\cdots,m_k\}$
  • Weak learning algorithm BaseClassifier.
  • Integer $T_k$, specifying the number of iterations.
Do for each $k=1,2,\ldots,K$:
  Initialize $w_1(i) = D_1(i) = 1/m_k,\ \forall i,\ i = 1,2,\cdots,m_k$
  If $k>1$, Go to Step 5, evaluate current ensemble on new data set
    $D_k$, update weight distribution;
  End If
  Do for $t = 1,2,\ldots,T_k$:
    1. Set $\boldsymbol{D_t} = \mathbf{w}_t \Big/ \sum_{i=1}^{m} w_t(i)$ so that $\boldsymbol{D_t}$ is a distribution.
    2. Draw training $TR_t$ and testing $TE_t$ subsets from $D_t$.
    3. Call BaseClassifier to be trained with $TR_t$.
    4. Obtain a hypothesis $h_t$ and calculate its error
         $\varepsilon_t = \sum_{i:h_t(\mathbf{x_i}) \neq y_i} D_t(i)$ on $S_k$.
       If $\varepsilon_t > ½$, discard $h_t$ and go to step 2.
    5. Call dynamically weighted majority voting (DWMV) to obtain
       the composite hypothesis
         $H_t = \arg\max_{y \in Y} \sum_{t:h_t(\mathbf{x})=y} DW_t(\mathbf{x})$
    6. Compute the error of the composite hypothesis
         $E_t = \sum_{i:H_t(\mathbf{x}_i) \neq y_i} D_t(i) = \sum_{i=1}^{m_k} D_t(i)[| H_t(\mathbf{x}_i) \neq y_i |]$
       If $E_t > ½$, discard $H_t$ and go to step 2.
    7. Set $B_t = E_t/(1-E_t)$, and update the weights:
         $w_{t+1}(i) = w_t(i) \times \begin{cases} B_t, & if\ \ H_t(\mathbf{x_i}) = y_i \\ 1\ , & otherwise \end{cases}$
         $= w_t(i) \times B_t^{1-[|H_t(\mathbf{x_i}) \neq y_i|]}$
  End
Call DWMV and output the final hypothesis:
  $H_{final}(\mathbf{x}) = \arg\max_{y \in Y} \sum_{k=1}^{K} \sum_{t:h_t(\mathbf{x})=y} DW_t(\mathbf{x})$
```

Fig. 1. Pseudocode for the modified Learn++ algorithm

$$H_t = \arg\max_{y \in Y} \sum_{t:h_t(\mathbf{x})=y} DW_t(\mathbf{x}) \tag{3}$$

where $DW_t(\mathbf{x})$ is the dynamic weight assigned to hypothesis h_t for the instance **x**. As described below, dynamic weights are determined by using Mahalanobis-distance based estimated likelihood of h_t to correctly classify the instance **x**. The composite error E_t made by H_t, that is, the performance of the entire ensemble constructed thus far, is then determined by summing up the distribution weights of all instances misclassified by the ensemble (step6).

$$E_t = \sum_{i:H_t(\mathbf{x}_i) \neq y_i} D_t(i) = \sum_{i=1}^{m_k} D_t(i)[| H_t(\mathbf{x}_i) \neq y_i |]. \tag{4}$$

Finally, the composite normalized error is determined as

$$B_t = E_t/(1-E_t), \quad 0 < E_t < \tfrac{1}{2}\ and\ 0 < B_t < 1 \tag{5}$$

and distribution weights are updated according to the ensemble performance (step 7).

$$w_{t+1}(i) = w_t(i) \times \begin{cases} B_t, & if\ \ H_t(\mathbf{x_i}) = y_i \\ 1\ , & otherwise \end{cases} \\ = w_t(i) \times B_t^{1-[|H_t(\mathbf{x_i}) \neq y_i|]}. \tag{6}$$

This expression reduces the weights of those instances correctly classified by the composite hypothesis H_t, by a factor of B_t, while the weights of the misclassified instances are kept unchanged. At t+1st iteration, after normalization of the weights in step 1, the probability of choosing previously correctly classified instances for TR_{t+1} is reduced, while that of misclassified instances is effectively increased.

This would be a logical place to pause and point out to some of the main difference between AdaBoost and Learn++. The distribution update rule in AdaBoost is based on the performance of the previous hypothesis [8], which focuses the algorithm on *difficult instances* with respect to different sampling of a given single database, whereas that of Learn++ is based on the performance of the entire ensemble [4], which focuses this algorithm on instances that carry *novel information* with respect to consecutive databases.

This becomes particularly critical when new database introduces instances from a previously unseen class. Since none of the previous classifiers in the ensemble has seen the instances from the new class, H_t initially misclassifies them, forcing the algorithm to focus on these instances that carry novel information. The procedure would not work nearly as efficiently, however, if the weight update rule were based on the performance of h_t only (as AdaBoost does) instead of H_t. This is because the training performance of the first h_t on instances from the new class is independent of that of the previously generated classifiers. Therefore, h_t is likely to correctly classify new class instances that it has just seen, but only at the time they are first introduced. This would cause the algorithm to focus on other *difficult to learn* instances, such as outliers, rather then the instances with novel information content.

Once T_k hypotheses are generated for each database D_k, the final hypothesis H_{final} can be obtained by combining all hypotheses by dynamically weighted majority voting, choosing the class that receives the highest total vote among all hypotheses:

$$H_{final}(\mathbf{x}) = \arg\max_{y \in Y} \sum_{k=1}^{K} \sum_{t:h_t(\mathbf{x})=y} DW_t(\mathbf{x})\ . \tag{7}$$

The intuition in using dynamically updated voting weights is as follows: if we knew which hypotheses would perform best ahead of time, we would give those hypotheses higher weights. We cannot have this information a priori, however, we can estimate which classifiers are more likely to correctly identify a given instance based on the location of that instance in the feature space with respect to the instances used to train individual classifiers. If an instance is *spatially close* – in a distance metric sense – to the training data used to train a classifier, then it is reasonable to expect that that classifier will perform well on the given instance.

We use the *class-specific Mahalanobis distance* metric to compute the distance between the training data and the unknown instance for each classifier. Classifiers whose training dataset are closer to the unknown instance are weighted higher. We note that previously seen data need not be stored in order to compute the desired distances, but only the means and covariance matrices of the training sets. We formalize the computation of these weights as follows:

Let us define TR_{tc} as the subset of TR_t, the training dataset used during the t^{th} iteration, to include only those instances that belongs to class c, that is,

$$TR_{tc} = \{\mathbf{x_i} \mid \mathbf{x_i} \in TR_t \ \& \ y_i = c\} \ni TR_t = \bigcup_{c=1}^{C} TR_{tc} \tag{8}$$

where C is the total number of classes. Class-specific Mahalanobis distance is then computed as,

$$M_{tc}(\mathbf{x}) = (\mathbf{x} - \mathbf{m_{tc}})^T \mathbf{C_{tc}}^{-1}(\mathbf{x} - \mathbf{m_{tc}}),\ c = 1,..,C \tag{9}$$

where $\mathbf{m_{tc}}$ is the mean and $\mathbf{C_{tc}}$ is the covariance metric of TR_{tc}. For any instance $\mathbf{x}$, the Mahalanobis distance based dynamic weight of the t^{th} hypothesis is then computed as

$$DW_t(\mathbf{x}) = \frac{1}{\min(M_{tc}(\mathbf{x}))},\ c = 1,..,C;\ t = 1,...,T \tag{10}$$

where T is the total number of hypotheses generated.

The Mahalanobis distance implicitly assumes that the underlying data distribution is Gaussian, which in general is not the case. Yet it is more informative then other distance metrics as it takes the data covariance into consideration, and provides promising results demonstrating its effectiveness.

III. Simulation Results

In this paper, we present simulation results of Learn++ with dynamic voting weight update along with Learn++ and Adaboost.M1 on one real world and two benchmark datasets. All results are given as 95% confidence interval obtained through 10-fold cross validation. To simulate incremental learning, the training is done in sessions, where only the most recently available database is shown to the algorithm during the current training session (TS).

A. Volatile Organic Compounds (VOC) Database

This database was generated from responses of six quartz crystal microbalances (QCMs) to various concentrations of five volatile organic compounds, Ethanol (ET), Octane (OC), Toluene (TL), Xylene (XL) and Trichloroethylene (TCE). The database was partitioned into four sets, S_1~S_3 for training, where each set introduces one new class, and TEST for validation. The data distribution is shown in Table 1. The base classifier used for all three algorithms was a single layer MLP with just enough hidden layer nodes and a rather tolerant error goal to make it a reasonably weak classifier for this database. Tables 2, 3, and 4 illustrate the percent training and generalization performances of Learn++ with dynamically updated voting weights (DUVW), original Learn++ and Adaboost.M1, respectively, on VOC data, after each training session, TS_1~TS_3.

TABLE 1. Data distribution for VOC database

Dataset	ET	OC	TL	TCE	XL
S₁	20	20	40	0	0
S₂	10	10	10	25	0
S₃	10	10	10	15	40
TEST	24	24	52	24	40

TABLE 2. DUVW- Learn++ performance on VOC Data

Dataset	TS₁	TS₂	TS₃
S₁	99.37~100.0	90.65~95.82	81.03~87.47
S₂	…	83.65~90.90	85.86~90.14
S₃	…	…	90.95~94.78
TEST	**59.71~60.81**	**68.39~70.90**	**86.91~88.60**

TABLE 3. Original Learn++ performance on VOC data

Dataset	TS₁	TS₂	TS₃
S₁	99.19~100.0	76.67~85.57	78.07~81.93
S₂	---	78.43~94.56	77.51~93.49
S₃	---	---	88.65~95.68
TEST	**61.70~62.62**	**67.82~71.50**	**84.68~88.46**

TABLE 4. Adaboost.M1 performance on VOC data

Dataset	TS₁	TS₂	TS₃
S₁	100.0	90.82~93.22	75.62~77.88
S₂	---	90.24~93.03	80.79~84.30
S₃	---	---	94.13~94.87
TEST	**61.21~61.95**	**71.67~72.38**	**81.65~82.58**

While all algorithms achieved incremental learning, Learn++ with dynamically updated voting weights performed best, just slightly better than the original version of Learn++, and significantly better than Adaboost.M1. It is also worth noting that the confidence interval of the modified Learn++ was also narrower than that of its predecessor, indicating less variability and increased stability in the performance of the modified algorithm.

Tables 2~4 also show some decline in training performances over three training sessions (on datasets S_1~S_3). This is expected due to stability-plasticity dilemma. We note, however that the loss of previously acquired knowledge – as measured by training data performance – is much less in the modified Learn++ then it is in others.

B. Wisconsin Breast Cancer (BC) Database

This database, originally created at The University of Wisconsin, Madison [16], was obtained from the UCI repository [17]. The database consists of nine features and a total of 683 instances from two classes of breast tumors: benign and malignant. The data distribution is shown in Table 5. The base classifier was again a MLP type neural network with similar characteristics as described earlier. Tables 6~8 present the 10-fold cross validation percent training and generalized performances.

TABLE 5. DATA DISTRIBUTION FOR BC DATA

Dataset	Benign	Malignant
S_1	100	85
S_2	104	70
TEST	**240**	**84**

TABLE 6. DUVW LEARN++ PERFORMANCE ON BC DATA

Dataset	TS_1	TS_2
S_1	93.97~97.92	93.76~96.18
S_2	---	94.35~95.76
TEST	**94.82~96.91**	**98~98.41**

TABLE 7. ORIGINAL LEARN++ PERFORMANCE ON BC DATA

Dataset	TS_1	TS_2
S_1	94.71~98.01	94.58~96.44
S_2	---	94.83~96.08
TEST	**96.34~97.12**	**97.57~98.17**

TABLE 8. ADABOOST.M1 PERFORMANCE ON BC DATA

Dataset	TS_1	TS_3
S_1	94.72~98.11	93.99~97.57
S_2	---	94.25~95.63
TEST	**94.95~97.76**	**97.66~98.51**

The training and generalization performances in Tables 6~8 indicate that all three algorithms do equally well in learning additional information if no new classes are introduced. In this application, all algorithms have acquired most of their knowledge from S_1 during the first training session, however, they were still able to extract incremental amount of new knowledge from the second dataset, S_2. The performance difference between the modified Learn++, original Learn++ and AdaBoost.M1 become less significant under such scenarios, where no new classes are introduced, or no substantial novel content is provided with the new database. This is expected, as the main difference between the original Learn++ and AdaBoost.M1 is the distribution update rule that is geared towards learning new classes. Similarly, the modification with the dynamic voting weights becomes more meaningful when different datasets cover substantially different portions of the feature space, which happens more drastically when either a new class is introduced, or the new instances carry substantial amount of novel information content.

C. Vehicle Silhouettes Database

Vehicle database was also obtained from the UCI repository [17]. This database consists of 18 features in 946 instances from four vehicle classes. This database is known to be challenging database, as typical performances on various algorithms on this database has reportedly been around 65~75% on non-incremental learning [17].

The vehicle database was divided into three training dataset $S_1 \sim S_3$ and one test dataset, *TEST*. The data distribution is shown in Table 9, which was specifically biased towards new classes. Tables 10~12 summarize 10-fold cross validation percent training and generalization performances of Learn++ with dynamic voting weight update, Learn++ and Adaboost.M1, respectively after each training session, $TS_1 \sim TS_3$. The base classifier used was again a single layer MLP type neural network, with similar characteristics as described above.

TABLE 9. DATA DISTRIBUTION FOR THE VEHICLE DATABASE

Dataset	Opel	Saab	Bus	Van
S_1	0	70	70	0
S_2	120	50	50	0
S_3	35	30	30	140
TEST	**57**	**67**	**68**	**59**

TABLE 10. DUVW LEARN++ PERFORMANCE ON VEHICLE

Dataset	TS_1	TS_2	TS_3
S_1	88.66~90.34	79.43~86.70	68.18~79.68
S_2	---	72.28~77.17	66.88~73.66
S_3	---	---	82.70~87.43
TEST	**47.00~49.09**	**52.81~55.55**	**71.79~75.46**

TABLE 11. ORIGINAL LEARN++ PERFORMANCE ON VEHICLE

Dataset	TS_1	TS_2	TS_3
S_1	89.60~92.60	68.81~88.76	60.94~76.78
S_2	---	50.15~70.76	49.24~64.03
S_3	---	---	78.02~86.92
TEST	**47.81~49.72**	**51.57~52.94**	**68.40~73.20**

TABLE 12. ADABOOST.M1 PERFORMANCE ON VEHICLE

Dataset	TS_1	TS_2	TS_3
S_1	67.20~90.80	55.86~91.57	59.07~83.21
S_2	---	37.59~62.69	40.67~52.88
S_3	---	---	40.63~80.90
TEST	**35.37~47.82**	**44.25~47.47**	**52.54~63.48**

The generalization (TEST) performances in Tables 10~12 indicate that the modified Learn++ has outperformed other two algorithms both in performance and in the confidence interval of the performance. Based on 10-fold cross validation, the generalization performance of the modified Learn++ was in 72~75% range, compared to 68~73% for original Learn++ and 52~63% for AdaBoost.M1. Furthermore, the modified Learn++ places itself much more favorably along the plasticity – stability spectrum, as it was able to retain significantly more of its previously acquired knowledge then the other algorithms.

IV. Discussion and Conclusions

In this paper, we presented a modified approach to weighted majority voting rule, where the classifiers are weighted dynamically for each instance, depending upon the estimated likelihood of the hypotheses to correctly classify the unknown instance. The intuitive idea behind this approach is that the classifier whose training dataset is closest to the given instance, has more information about that particular instance and therefore is more likely to classify that instance correctly.

Simulation results indicate that all three algorithms are capable of incremental learning; however, the results were most favorable and promising for the modified Learn++ using dynamically updated voting weights. We note that the generalization performances obtained by the modified Learn++ during *incremental learning* were very similar, if not better, then the generalization performances obtained by several other algorithms on these datasets when used in a *non-incremental learning* setting as reported in [16]. Learning in a non-incremental setting allows the entire data to be made available to the algorithm at once, which is a much simpler problem.

The modified Learn++ algorithm exhibited not only a better generalization performance, but also a significantly narrower confidence interval. The improved confidence interval is in fact worth attention. This is because a narrower confidence interval indicates improved stability and robustness, qualities of considerable concern in incremental learning. In particular, improved generalization performance coupled with a narrower confidence interval is a satisfying outcome, since this combination places the modified Learn++ very favorably on the stability-plasticity spectrum.

We also run all algorithms multiple times under several other scenarios, such as changing the order in which the training data are presented, and changing the base classifier training parameters (such as number of hidden layer nodes, error goal, etc.). We have found out that all algorithms are robust to the order in which the datasets are presented, as well as to reasonable modifications in the training parameters. Also, none of the algorithms suffer from catastrophic forgetting, since previously generated classifiers are retained. Loss of some information is inevitable due to the stability-plasticity dilemma while new information is being learned. However, this loss of previously acquired knowledge was very marginal with the modified Learn++, but most prominent with AdaBoost, when the data introduced significant amount of novel information content, such as a new class.

We conclude by restating that in applications where the additional information content is minimal, the performance differences between the algorithms become less significant. The promising results of the modified Learn++ with dynamically updated voting weights are most meaningful and beneficial when the algorithm is used under the scenarios for which it is specifically designed, that is, when the additional data provide significant novel information content.

Acknowledgement

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] M. McCloskey and N. Cohen, "Catastrophic interference in connectionist networks: the sequential learning problem," in *The Psychology of Learning and Motivation,* G.H. Bower, ed., vol. 24, pp. 109-164, Academic Press, San Diego, 1989.

[3] R. French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences*, vol. 3, no.4, pp. 128-135, 1999.

[4] R. Polikar, L. Udpa L, S. Udpa, and 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.

[5] R. Polikar, J. Byorick, S. Krause, A. Marino, and M. Moreton, "Learn++: A classifier independent incremental learning algorithm for superv. neural netw.," *Proc. of Int. Joint Conf. on Neural Networks.*, vol. 2, pp. 1742-1747, Honolulu, HI, 2002.

[6] R. Polikar, L. Udpa L, S. Udpa, and V. Honavar, "An incremental learning algorithm with confidence estimation for automated identification of NDE signal**s**," *IEEE Trans.Ultraso., Ferro., Freq. Cont.,* vol. 51, no. 8, pp. 990-1001, 2004.

[7] L. Hansen and P. Salamon, "Neural network ensembles," *IEEE Trans. on PAMI*, vol. 12, no. 10, pp. 993-1001, 1990.

[8] Y. Freund and R. Schapire, "A decision theoretic generalization of online learning and an appli. to boosting," *Comp. and System Sciences*, vol. 57, no. 1, pp. 119-139, 1997.

[9] N. Littlestone and M. Warmuth, "Weighted majority algorithm," *Infor. and Comput.,* vol. 108, pp. 212-261, 1994.

[10] M.I. Jordan and R.A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm," *Neural Computation*, vol. 6, no. 2, pp. 181-214, 1994.

[11] 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.

[12] L. Breiman, "Combining predictors," *Combining Artificial Neural Nets,* A.Sharkey, ed., pp. 31-50, NY: Springer 1999.

[13] T. Dietterich, "Ensemble methods in machine learning," *Proc. 1st Int. Wkshop on Mult. Class. Sys., LNCS* , J. Kitler, F. Roli, ed., vol. 1857, pp.1-15, NY, Springer. 2000.

[14] 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, NY: Springer, 2002.

[15] T. Windeatt and F. Roli (eds), In *Proc. 4th Int. Workshop on MCS, LNCS*, vol. 2709, NY, Springer, 2003.

[16] W. H. Wolberg and O.L. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology," *Proc. of the National Academy of Sciences*, vol. 87, pp 9193-9196, 1990.

[17] C.L. Blake and C.J. Merz, Univ. of California, Irvine, Repository of Machine Learning Databases at Irvine, CA.