# INCREMENTAL LEARNING OF NDE SIGNALS WITH CONFIDENCE ESTIMATION

*Robi Polikar*

*Department of Electrical and Computer Engineering*
*Rowan University, Glassboro, NJ 08028, USA*

**Abstract**. An incremental learning algorithm, Learn++, is introduced, for learning additional information from new data, even when new data include examples of previously unseen classes. Learn++ takes advantage of synergistic generalization performance of an ensemble of simple classifiers, each trained with a strategically chosen subset of the training database. As new data become available, new classifiers are generated, which are then combined through weighted majority voting. The weights are determined based on the estimated likelihood of each classifier to correctly classify an instance of unknown class. The voting procedure also allows Learn++ to estimate the confidence level in its own decision.

## INTRODUCTION

An increasing number of nondestructive evaluation (NDE) applications resort to pattern recognition and machine learning algorithms for automated classification and characterization of NDE signals. Applications of such systems include defect identification in aircraft engines and wheels [1], in tubings and pipings of nuclear power plants [2, 3], in artificial heart valves, or even in concrete sewer pipelines [4]. In any automated signal classification system, the performance of the resulting classifier relies heavily on the availability of a representative set of training examples. In many practical applications, however, acquisition of a representative training data is expensive and time consuming. Consequently, it is not uncommon for such data to become available in small batches over a period of time. This is particularly true for NDE applications, where large volumes of data need to be analyzed. For example, in nuclear power plants, data are collected from various tubings or pipings during different outage periods, and new types of defect and geometry indications can be discovered in aging components. Classification algorithms developed using previously collected databases may then become inadequate in successfully identifying new types of indications.

In such settings, it is necessary to update an existing classifier in an incremental fashion to accommodate new data without compromising classification performance on old data. Majority of the popular signal classification algorithms that are commonly used in NDE applications, however, do not allow incremental learning of additional data particularly when the new data introduces additional classes. One common approach taken by is therefore involves discarding the existing classifier, and retraining a new classifier using a combination of previously used data and the new data. This results in all previous learning to be lost, a

CP615, *Review of Quantitative Nondestructive Evaluation Vol. 21*, ed. by D. O. Thompson and D. E. Chimenti

phenomenon known as *catastrophic forgetting*. Furthermore, this approach will not even be feasible if the previously used data is no longer available, a scenario that arises often.

Another issue that is of interest in using automated signal classification systems is the confidence of such systems in their own decisions. Often times, these systems do make mistakes, by either missing a defect or incorrectly classifying a geometrical or benign indication as a defect (false alarm). Both types of mistakes have dire consequences: missing defects can cause unpredicted and possibly catastrophic failure of the material, whereas a false alarm can cause unnecessary and premature part replacement, resulting in serious economic loss. A classification system that can estimate its own confidence would be able to flag those cases where the classification may be incorrect, and such cases can then be further analyzed. An algorithm that can

- learn from new data in the absence of previously used data;
- retain the previously acquired knowledge;
- accommodate new classes; and
- estimate the confidence in its own classification

would therefore be of great benefit to NDE community. In this paper, we present an improved version of the Learn++ algorithm, which satisfies the above-mentioned criteria. The original Learn++ algorithm, which was presented in [5, 6], achieves incremental learning by generating an ensemble of simple classifiers for each additional database, which are then combined by a weighted majority voting algorithm. The new version, presented here adds improved performance using a different voting mechanism, as well as the capability of the estimating the decision confidence. The algorithm was tested on a variety of databases: results involving ultrasonic weld inspection signals, are presented in this paper. Applications to other databases can be found in [7].

## INCREMENTAL LEARNING

### Ensemble of Classifiers

Incremental learning has been a topic of active research interest in machine learning and artificial intelligence. Several terms, such as on-line learning, life long learning, evolutionary learning, and constructive learning, have been used interchangeably with incremental learning, and therefore, several variations of this problem have been addressed in the literature [7]. Such variations include retraining using a combination of original training data and the new data, incrementally growing or pruning network architectures, restricted modification of weights for misclassified signals, as well as on-line learning, where the training is carried out on an instance-by-instance basis. In this paper, we define an incremental learning algorithm as one that is capable of learning additional information from new data, which may include new classes, without forgetting prior knowledge and without requiring access to previously used data.

Learn++ is inspired by Schapire's *adaptive boosting (AdaBoost)* algorithm [8], which was originally proposed for improving the accuracy of weak learning algorithms. The idea is to generate an ensemble of weak classifiers using different distributions of the training data, followed by a majority voting of the outputs of the weak classifiers to obtain the final hypothesis. Combining weak classifiers take advantage of the so-called *instability* of the weak classifier, which causes the classifiers to construct sufficiently different decision surfaces for minor modifications in their training datasets. Littlestone *et al.* have shown that the *weighted majority algorithm*, which assigns weights to different hypotheses based on an error criterion to construct a compound hypothesis, performs better than any of the individual hypotheses [9]. They also showed that the error of the compound hypothesis is closely linked to the error bound of the best hypothesis. In essence, both AdaBoost and Learn++ employ a weighted combination of a group of classifiers, rather than using just one classifier as

conventional classification algorithms do. The main difference between AdaBoost and Learn++ is that the distribution update rule for AdaBoost is optimized for improving the accuracy of the classifier on a given database, whereas the distribution update rule for Learn++ is optimized for learning new information, in particular, new classes. The original Learn++ also used the weighted majority voting to combine the classifiers, where the voting weight of each classifier was determined based on the individual performance of the classifier on its own training dataset. The new version of Learn++ uses the Mahalonobis distance of the unknown instance to the training datasets to estimate which classifier is more likely to classify the given instance correctly, and determines the voting weights accordingly.

In the next section we provide the new version of the Learn++ algorithm, along with the intuitive confidence estimation mechanism based on the voting mechanism. The results on selected ultrasonic weld inspection database are provided next, followed by concluding remarks and directions for future research. An excellent overview of ensemble of classifiers can be found in [10], whereas a review of comparing ensemble of classifiers with other types of learners can be found in [11].

## Learn++ for Incremenal Learning

Figure 1 illustrates the new Learn++ algorithm, which runs iteratively for each new dataset $\mathcal{D}_k$ that becomes available to the classifier. The inputs to Learn++ include a sequence of $m$ training instances with corresponding target classes, $S=[(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),\ldots,(\mathbf{x}_m,y_m)]$, weak learning algorithm, **WeakLearn**, and an integer $T_k$ specifying the number of iterations. For each dataset $\mathcal{D}_k$, Learn++ starts by initializing a distribution function $D_1(i)=1/m,\ i=0,1,\ldots,m$ on current dataset, such that the training instances which will be drawn according to this distribution will have equal likelihood to be selected into the first training set (Step 1). Learn++ then enters into an iterative loop, where at each iteration $t$, a new hypothesis is generated. During the $t^{th}$ iteration, Learn++ selects a training dataset $TR_t$ and a test dataset $TE_t$ according to the current distribution $D_t$ in step 2. In step 3, the weak learning algorithm WeakLearn is trained using the training dataset $TR_t$. A classification rule, $h_t$, is obtained as the $t^{th}$ hypothesis in step 4. Also in this step, $\varepsilon_t$, the error of $h_t$ is obtained by evaluating the hypothesis on all instances in $TR_t$ and $TE_t$. If this error is greater than ½, current $h_t$ is discarded, and the algorithm returns to step 1 to select a different training dataset. In step 5, Learn++ computes the mean and variance of the current training dataset, to be used in determining the Mahalanobis distances. Learn++ then calls the Mahalonobis weighted majority (MWM) algorithm to compute the composite hypothesis, $H_t$. The MWM computes the total vote each class receives, from all previous $t$ hypotheses. Each vote is weighted by $MW_t$ the inverse of the shortest Mahalonobis distance of the current instance $\mathbf{x}$ with all $TR_{tc}$, subsets of $TR_t$ belonging to class $c$, that have been seen until that iteration:

$$M_{tc} = (\mathbf{x}-\mathbf{m}_{tc})^T \mathbf{C}_{tc}^{-1} (\mathbf{x}-\mathbf{m}_{tc}) \tag{1}$$

$$MW_t = \frac{1}{\min(M_{tc})} \quad c = 1,2,\cdots,C \tag{2}$$

where $\mathbf{m}_{tc}$ is the mean, and $\mathbf{C}_{tc}$ is the covariance matrix of $TR_{tc}$

Note that this computation requires only the mean vectors and covariance matrices of the datasets to be saved, but not the datasets themselves. Also, note that this is in contrast to the original Learn++ algorithm, where the voting weights were based on the classification performances of the individual hypotheses on their own training data.

The combination rule ensures that hypotheses that were generated using data similar to the current instance being evaluated are weighted more heavily than the others. Note that voting weights are therefore updated dynamically for each instance. The class that receives the highest total vote becomes the classification rule of the $t^{th}$ composite hypothesis $H_t$.

**Algorithm Learn++ with Mahalanobis Weighted Majority Voting**

**Input**: For each dataset drawn from $\mathcal{D}_k$ $k=1,2,\ldots,K$

- Sequence of $m$ training examples $S=[(x_1,y_1),(x_2,y_2),\ldots,(x_m,y_m)]$.
- Weak learning algorithm **WeakLearn** (MLP).
- Integer $T_k$, specifying the number of iterations.

**Do for** $k=1,2,\ldots,K$:

**Initialize** $w_1^i = D(i) = 1/m$, $\forall i$, unless there is prior knowledge to select otherwise.

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

1. Set $\mathbf{D}_t = \mathbf{w}_t \Big/ \sum_{i=1}^{m} w_t(i)$ so that $\mathbf{D}_t$ is a distribution.
2. Randomly choose training $TR_t$ and testing $TE_t$ subsets according to $\mathbf{D}_t$.
3. Call **WeakLearn**, providing it with $TR_t$.
4. Get back a hypothesis $h_t : X \rightarrow Y$, and calculate the error of $h_t$ : $\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$ on $TR_t + TE_t$. If $\varepsilon_t > 1/2$, set $t = t-1$, discard $h_t$ and go to step 2.
5. Compute the variances and means of the datasets used, and call Mahalanobis weighted majority, to obtain composite hypothesis $H_t = \arg\max_{y\in Y} \sum_{t:h_t(x)=y} MW_t$, where $MWt$ is the Mahalanobis weight of $t^{th}$ hypothesis.
6. Compute the overall error $E_t = \sum_{i:H_t(x_i)\neq y_i} D_t(i) = \sum_{i=1}^{m} D_t(i)\left[\left[ H_t(x_i) \neq y_i \right]\right]$. If $E_t > 1/2$, set $t = t-1$, discard $H_t$ and go to step 2.
7. Set $B_t = E_t/(1-E_t)$, and update the weights of the instances:

$$w_{t+1}(i) = w_t(i) \times \begin{cases} B_t, & \text{if } H_t(x_i) = y_i \\ 1, & \text{otherwise} \end{cases}$$
$$= w_t(i) \times B_t^{1-\left[\left[H_t(x_i)\neq y_i\right]\right]}$$

**Call** Mahalanobis weighted majority on all hypotheses generated so far and

**Output** the final hypothesis: $H_{final} = \arg\max_{y\in Y} \sum_{k=1}^{K} \sum_{t:h_t(x)=y} MW_t$.

**FIGURE 1**. Algorithm Learn++ with Mahalonobis weighted majority voting.

The error $E_t$ and the normalized error $B_t$ of the composite hypothesis are then computed in step 6. Finally, in step 7, Learn++ updates the distribution $D_t$ according to the performance of $H_t$. The distribution update rule decreases the weight of all instances that are correctly classified by $H_t$, such that they are less likely to be selected into the next training dataset.

Therefore, as the algorithm proceeds, increasingly difficult examples are selected into the training dataset. This procedure allows rapid learning of new data, since only those instances of the new dataset that are misclassified by the classifier are used for further training the classifier. After a pre-specified number of hypotheses are generated, Learn++ computes the final hypothesis, using a MWM voting on all hypotheses generated up to that point.

Finally, the voting mechanism employed in Learn++ hints a simple procedure for estimating the confidence of the algorithm in its own classification decisions. In essence, if the majority of the (weighted) hypotheses agree on the class of a particular instance, we can interpret this outcome as a high confidence decision. If, on the other hand, the individual hypothesis votes are distributed equally among different classes, the final decision can be

interpreted as a low confidence decision. To formalize this approach, let us assume that there are a total of $T$ hypotheses generated in $K$ training sessions for classifying instances into one of $C$ classes.

We can then define $\xi_c$, the total vote that class $c$ receives, as

$$\xi_c = \sum_{t:h_t(x)=c} MW_t \quad t=1,\cdots,T, \quad c=1,\cdots,C. \tag{3}$$

The final classification will then be the class for which $\xi_c$ is maximum. Normalizing the votes received by each class

$$\xi_c = \xi_c \Big/ \sum_{c=1}^{C} \xi_c \tag{4}$$

allows us to interpret $\xi_c$ as a measure of confidence of the decision on a 0 to 1 scale, with 1 corresponding to maximum confidence and 0 to no confidence. It should strictly be noted however that normalized $\xi_c$ values do *not* represent the *accuracy* of the results, nor are they related to the statistical definition of confidence intervals determined through hypothesis testing. This is merely a measure of the confidence of the algorithm in its own decision. Keeping this distinction in mind, we can heuristically define the following ranges: $0.0 < \xi_c < 0.6$ very low confidence, $0.6 < \xi_c < 0.7$ low confidence, $0.7 < \xi_c < 0.8$ medium confidence, $0.8 < \xi_c < 0.9$ high confidence and $0.9 < \xi_c < 1$ very high confidence.

The algorithm was implemented on a database of ultrasonic weld inspection signals obtained from nuclear power plant pipes. The database is briefly described first followed by results on incremental learning and confidence estimation.

## ULTRASONIC INSPECTION OF NUCLEAR POWER PLANT PIPES

Welding regions are often susceptible to various kinds of defects due to imperfections introduced into the material during the welding process. In nuclear power plant pipes, such defects manifest themselves in the forms of intergranular stress corrosion crackings, usually in an area immediately neighboring the welding region. This region in known as the heat affected zone. Such cracks can be detected by using ultrasonic techniques. However, also in vicinity of this area, there are often other type of reflectors, including counterbores and weld roots, which are considered as geometric reflectors. Counterbores and weldroots do not pose any threat to the structural integrity of pipe, however, they often generate signals that are very similar to those generated by cracks. Accurate detection and identification of cracks from their ultrasonic footprints is therefore crucial, since they can eventually cause structural damages if remedial actions are not taken. Figure 2 conceptually illustrates the ultrasonic testing procedure, used to generate the database used in this study. 1 MHz ultrasonic transducers were used. Figure 3 illustrates typical signals from each type of reflector.

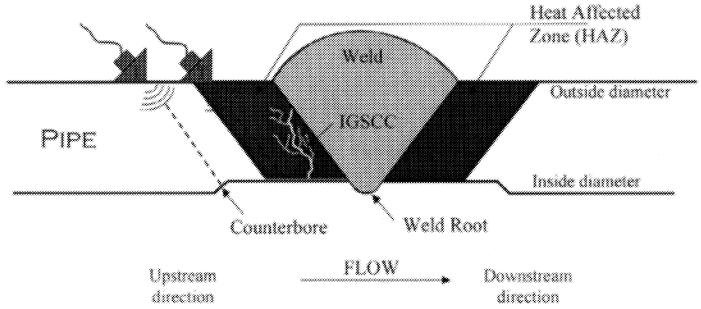


**FIGURE 2.** Ultrasonic testing of nuclear power plant pipes.

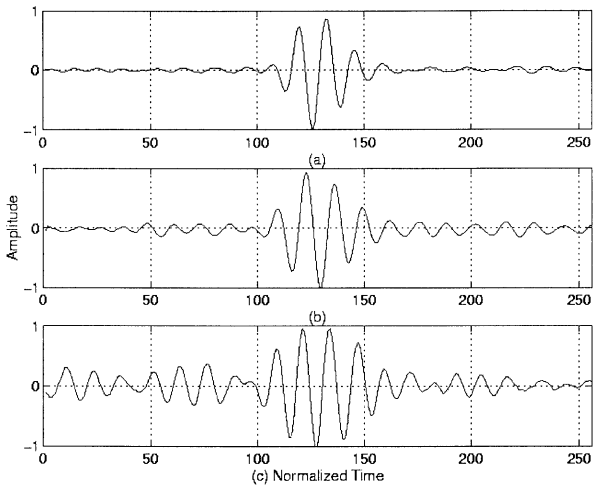


**FIGURE 3.** Sample signals from (a) crack, (b) counterbore and (c) weld root.

## RESULTS AND DISCUSSIONS

The goal of the classification algorithm in this application is the identification of three different types of indicators, namely, crack, counterbore and weldroot from the discrete wavelet transform coefficients of the ultrasonic A-scans. Three training databases $S_1$~$S_3$ of 300 instances each, and a validation database, *TEST,* of 487 instances were acquired from the above described system. 48 - point DWT coefficients were computed for each A-scan to be used as feature vectors. The training databases were made available to the algorithm at different times. During each of the three training sessions, only one of the training databases was made available to the algorithm to test the incremental learning capability of Learn++ on this database. The weak learning algorithm was a single hidden layer MLP of 30 nodes, with a large mean square error goal of 0.1. Note that MLPs can be used as weak learners, when their architecture is kept small, and their error goal is kept high, with respect to the complexity of the classification problem.

Table 1 presents the results, where rows indicate the classification performance of Learn++ on each of the databases after each training session ($TS_i$, $i=1,2,3$). The numbers in parentheses in the first row indicate the number of weak classifiers generated in each training session. We note that the performance on the validation dataset *TEST* improved steadily as new databases were introduced, demonstrating the incremental learning ability of the algorithm. Also, note that the algorithm was able to maintain its classification performance on the previous datasets, after training with additional datasets. This shows that previously acquired knowledge was not lost.

**TABLE 1.** Classification performance of Learn++ on the weld inspection database.

| | **$TS_1$ (6)** | **$TS_2$ (10)** | **$TS_3$ (14)** |
|---|---|---|---|
| $S_1$ | 95.7% | 95% | 94.3% |
| $S_2$ | --- | 95% | 95.3% |
| $S_3$ | | --- | 96.1% |
| ***TEST*** | **81.9%** | **91.7%** | **95.6%** |

As a comparison, the classification performance of a strong learner, with two hidden layers of 30 and 7 nodes, respectively, and an error goal of 0.0005, was also around 95%, however, the entire training database (900 instances) were used to train this classifier. We

therefore conclude that Learn++, by only seeing a fraction of the training database at a time in an incremental fashion, can perform as good as (or better then) a strong learner that is trained with the entire database at once.

The algorithm's confidence in each decision was also computed as described earlier. Table 2 lists a representative subset of the classification results and confidence levels on the validation dataset after each training session. A number of interesting observations can be made from Table 2, which is divided into four sections. The first section shows those instances that were originally misclassified after the first training session, but were correctly classified after the second or third sessions. There were 66 such cases (out of 487 in the TEST dataset). Many of these were originally misclassified with rather strong confidences. Note that during the next two training sessions, not only their classification was corrected, but also the confidence on the classification improved as well.

The second section of the table shows those cases on which the confidence improved with training. Majority of the instances (396 out of 487) were of this case. The third section shows examples of those cases, which were still misclassified at the end of three training sessions, but the classification confidence *decreased* with training. Note that the confidence was very high after the first training session, which decreased to very low after the third training session. There were 21 such instances, and these are the instances flagged by the algorithm to be reevaluated. Finally, the fourth section shows the only four instances where the algorithm either increased its confidence in misclassification or decreased its confidence in correct classification. These are considered as isolated instances (or outliers), since there were only four such instances in the entire database.

**TABLE 2.** Sample confidences on the TEST dataset for each training session.

| Instance # | True Class | Training Session 1 | | Training Session 2 | | Training Session 3 | |
|---|---|---|---|---|---|---|---|
| | | Class | Conf | Class | Conf | Class | Conf |
| Misclassification Corrected with Improved Confidence (66) | | | | | | | |
| 25 | Crack | Cbore | 0.69 | Crack | 0.91 | Crack | 0.96 |
| 144 | Crack | Cbore | 0.54 | Crack | 0.86 | Crack | 0.91 |
| 153 | Crack | Cbore | 0.35 | Crack | 0.54 | Crack | 0.76 |
| 177 | Cbore | Crack | 0.81 | Crack | 0.55 | Cbore | 0.71 |
| 206 | Crack | Root | 0.43 | Crack | 0.51 | Crack | 0.71 |
| 267 | Cbore | Crack | 0.52 | Cbore | 0.86 | Cbore | 0.96 |
| 286 | Crack | Cbore | 0.7 | Crack | 0.54 | Crack | 0.84 |
| 289 | Root | Crack | 0.92 | Crack | 0.84 | Root | 0.76 |
| 308 | Root | Cbore | 0.69 | Root | 0.47 | Root | 0.87 |
| 323 | Crack | Cbore | 0.82 | Cbore | 0.79 | Crack | 0.76 |
| 352 | Crack | Root | 0.81 | Crack | 0.8 | Crack | 0.89 |
| 354 | Root | Crack | 0.87 | Crack | 0.64 | Root | 0.79 |
| 438 | Crack | Cbore | 0.57 | Crack | 0.76 | Crack | 0.92 |
| 454 | Cbore | Root | 1 | Root | 1 | Cbore | 0.58 |
| Improved Confidence in Correct Classification (396) | | | | | | | |
| 67 | Cbore | Cbore | 0.66 | Cbore | 0.94 | Cbore | 0.96 |
| 313 | Crack | Crack | 0.6 | Crack | 0.73 | Crack | 0.88 |
| 321 | Cbore | Cbore | 0.47 | Cbore | 0.87 | Cbore | 0.93 |
| 404 | Root | Root | 0.59 | Root | 0.73 | Root | 0.96 |
| Reduced Confidence in Misclassification (21) | | | | | | | |
| 261 | Crack | Cbore | 1 | Cbore | 1 | Cbore | 0.54 |
| 440 | Cbore | Root | 1 | Root | 0.85 | Root | 0.66 |
| 456 | Cbore | Root | 0.65 | Cbore | 0.52 | Crack | 0.55 |
| Utterly Confused Classifier (4) | | | | | | | |
| 3 | Root | Root | 0.49 | Crack | 0.55 | Crack | 0.59 |
| 45 | Crack | Crack | 0.78 | Crack | 0.61 | Crack | 0.53 |
| 78 | Cbore | Crack | 0.67 | Cbore | 0.52 | Crack | 0.55 |
| 93 | Root | Root | 0.94 | Crack | 0.58 | Crack | 0.58 |

A second database obtained from a similar experimental setup, but with four types of defects, namely, crack, lack of fusion, porosity and slag, was also used to evaluate Learn++ on an incremental learning problem that involved addition of previously unseen classes. In this case, the database was also divided into three datasets, however, second and third datasets introduced additional classes. Learn++ was able to learn the new information, as well as the new classes with high classification accuracy on a validation dataset. The results of this experiment have been reported in [5].

## CONCLUSIONS AND FUTURE WORK

We have introduced Learn++, as an incremental learning algorithm for supervised neural networks. The algorithm is also capable of estimating its own confidence through a simple voting mechanism. The feasibility of the approach has been validated on an inherently difficult database of ultrasonic weld inspection signals.

Future work includes optimizing the distribution update rule for faster and more robust training, and testing the versatility of Learn++ with different types of classifiers used as weak learning algorithms.

## REFERENCES


1. Nawapak, E.A., Udpa, L., Chao, J., "Morphological Processing for Crack Detection in Eddy Current Images of Jet Engine Disks," in *Review of Progress in Quantitative Nondestructive Evaluation*, edited by Thompson, D. O., and Chimenti, D.E., Plenum Press, New York, 1999, pp. 751-758.
2. Polikar, R., Udpa, L., Udpa, S.S., and Taylor, T., *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control* **45**, 614-625, (1998).
3. Ramuhalli, P., Udpa, L., Udpa, S.S., *Materials Evaluation* **58**, 65-69, (2000).
4. Mandayam S., Jahan K., Cleary D.B., "Ultrasound inspection of wastewater concrete pipelines – signal processing and defect characterization," in *Review of Progress in Quantitative Nondestructive Evaluation*, edited by Thompson, D. O., and Chimenti, D.E., AIP Press, New York, 2001, pp. 1210-1217.
5. Polikar R., Udpa L., Udpa S.S., "Incremental learning of ultrasonic weld inspection signals," in *Review of Progress in Quantitative Nondestructive Evaluation*, edited by Thompson, D. O., and Chimenti, D.E., AIP Press, New York, 2001, pp. 603-610.
6. Polikar, R., Udpa, L., Udpa, S.S., Honavar, V., "Learn++: An incremental learning algorithm for multilayer perceptron networks," in *Proceedings of IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, pp. 3414-3417.
7. Polikar, R., *Algorithms for enhancing pattern separability, optimum feature selection and incremental learning with applications to gas sensing electronic nose systems*, Ph.D. dissertation, Iowa State University, 2000, Chapter 6.
8. Freund, Y., and Schapire, R., *Computer and System Sciences* **57**, 119-139 (1997).
9. Littlestone, N., and Warmuth, M., *Information and Computation* **108**, 212-261, (1994).
10. Ji, C., and Ma, S., *IEEE Proceedings,* **87**, 1519-1535, (1999).
11. Dietterich, T.G., *AI Magazine*, **18,** 97-136, (1997).