Incremental Learning from Unbalanced Data

Michael Muhlbaier
Department of Electrical and Computer Engineering
Rowan University
Glassboro, NJ 08028 USA
E-Mail: m.muhlbaier@ieee.org

Apostolos Topalis
Department of Electrical and Computer Engineering
Rowan University
Glassboro, NJ 08028 USA
E-Mail: a.topalis@ieee.org

Robi Polikar
Department of Electrical and Computer Engineering
Rowan University
Glassboro, NJ 08028 USA
E-Mail: polikar@rowan.edu

Abstract— **An ensemble based algorithm, Learn++.MT2, is introduced as an enhanced alternative to our previously reported incremental learning algorithm, Learn++. Both algorithms are capable of incrementally learning novel information from new datasets that consecutively become available, without requiring access to the previously seen data. In this contribution, we describe Learn++.MT2 which specifically targets incrementally learning from distinctly unbalanced data, where the amount of data that become available varies significantly from one database to the next. The problem of unbalanced data within the context of incremental learning is discussed first, followed by a description of the proposed solution. Initial, yet promising results indicate considerable improvement on the generalization performance and the stability of the algorithm.**

I. Introduction

We have previously introduced Learn++, an incremental learning algorithm designed to learn novel information content from an additional set(s) of data that become available after a classifier has already been trained with the original data. Applications that require learning under such settings arise often in practice: it is not unusual for data to become available through several data collection sessions, which may be several months or years apart from each other. Such scenarios require a classification system to be incrementally updated – as new data become available – where the classifier needs to learn the novel information content without forgetting the previously acquired knowledge. Grossberg showed that this problem faces the stability-plasticity dilemma [1]: stability allows a classifier to retain the acquired knowledge, whereas plasticity allows a classifier to acquire new knowledge. Unfortunately, these two properties are typically at odds with each: the more stable a classifier the less plastic it is, and vise versa. For example, the multilayer perceptron (MLP) and radial basis function networks – ubiquitously used in real-world applications – are very stable classifiers and hence unable to learn incrementally from new data. The approach typically taken involves discarding the existing classifier, combining the old and new data, and retraining the classifier using the combined data. Discarding the existing classifier, however, causes the previously learned information to be lost, a phenomenon known as catastrophic forgetting [2]. Furthermore, this approach is not even feasible, if the original dataset is no longer available.

Learn++ was proposed as an incremental learning algorithm that exhibits a fine balance across the stability – plasticity spectrum: it is capable of learning from new data, while substantially retaining previous knowledge and without requiring access to the previously used data, even when the new dataset includes instances from previously unseen classes [3]. The algorithm, inspired by AdaBoost, takes advantage of the synergistic learning ability of an ensemble of classifiers. Unlike AdaBoost, which seeks to improve the generalization performance of a weak classifier [4], Learn++ aims to incrementally learn the newly available information. More specifically, each classifier generated by Learn++ is trained on a subset of the current training dataset. The instances of each subset are drawn according to an iteratively updated distribution that is strategically biased towards those instances that carry novel information. The relative performance of each classifier on its training data then determines its voting weight to be used in weighted majority voting [5], where the ensemble chooses the class that receives the highest total vote from individual classifiers. As new data become available, Learn++ generates additional classifiers, until the ensemble learns the novel information. Since no classifier is discarded, previously acquired knowledge is not lost.

II. Unbalanced Data

While Learn++ works rather well on a broad spectrum of applications [6], there is room for improvement. First, determining when the algorithm should stop is tricky, as the performance of Learn++ starts deteriorating after a certain number of classifiers are generated. One may be suspicious of overfitting, which can be reduced by using a validation dataset to determine when the algorithm should stop. However, we believe that the problem is not just overfitting, since the performance deterioration appears when – and only when – the algorithm is run on addi-

0-7803-8359-1/04/$20.00 ©2004 IEEE

tional datasets. No significant performance deterioration is observed while Learn++ is generating its first set of classifiers with the original data. We suspect that the problem is – at least in part – due to unbalanced data.

An interesting problem in the incremental learning setting is the issue of unbalanced data, which we define as the discrepancy in the *cardinality* of each dataset used in incremental learning. If one dataset has substantially more data than the other, this can unfairly bias the ensemble decision towards the data with the lower cardinality. This is because, the voting weights of each classifier is determined solely by its performance on its respective training data. Even though the cardinality of a given dataset may be small, the classifier may perform well on its own limited training dataset, and therefore receive a high voting weight. This classifier is most likely to perform poorly – relative to other classifiers generated with larger cardinality data – on the unseen instances, since it was trained on limited data.

In the absence of any other information, and under the generally valid assumptions that (i) no instance is repeated and (ii) the noise distribution remains relatively unchanged among datasets, it is reasonable to believe that the dataset that has more instances carries more information. Classifiers generated with such data should therefore be weighted more heavily.

It is not unusual to see major discrepancies in the cardinalities of datasets that subsequently become available. Consequently, in any ensemble based learning algorithm that employs a classifier combination scheme, the cardinality of each dataset should be taken into consideration.

An unbalanced data need not be caused simply due to discrepancy among dataset cardinalities, but may also be due to relative cardinalities of individual classes within the training data, a quantity often described as class prior probabilities. While class priors appear conspicuously within the Bayesian setting, they are not as heavily utilized in many other algorithms. Commonly used ensemble combination schemes, such as voting, sum or product based combination, often do not take class priors into consideration [7,8]. Bibliography on ensemble systems, incremental learning approaches, and their applications can be found in and within the references of [3, 6 ~ 15].

In this contribution, we propose a set of modifications to address both aspects of unbalanced data described above. We present the algorithm and some preliminary results on two benchmark database and one real world classification problem. While, the approach is described specifically for Learn++, it is nevertheless quite general and can be easily adapted to any ensemble based algorithm.

II. LEARN++.MT2

The primary novelty in Learn++.MT2 is the way by which the voting weights are determined. Learn++.MT2 attempts to addresses the unbalanced data problem by keeping track of the number of instances from each class with which each classifier is trained. Similar to Learn++, each classifier is first given a weight based on its performance on its own training data; however, this weight is later adjusted according to its *class conditional weight factor*. For each classifier, this is the ratio of instances from a particular class used for training that classifier, to the number of instances from that class used for training all classifiers thus far within the ensemble. The pseudocode of the entire algorithm is given in Figure 1.

For each dataset ($\mathcal{D}_k$) that becomes available, the inputs to the algorithm are (i) a sequence of m training data instances x_i along with 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 during the k^{th} training session. For the first database ($k=1$), a data distribution (D_1) – from which training instances will be drawn - is initialized to be uniform, making the probability of any instance being selected equal. The number of instances from each class $c \in \{1,\ldots,C\}$ in $\mathcal{D}_1$ is stored in N_c. If $k>1$ then a distribution initialization sequence re-initializes the data distribution (the IF block in Figure 1). The variable eN_c holds the current value of N_c which is then updated as the sum of all class-c instances contained in $\mathcal{D}_1$ through $\mathcal{D}_k$.

The algorithm then adds T_k classifiers to the ensemble starting at $t=eT_k+1$ where eT_k is the number of classifiers that currently exist in the ensemble. For each iteration t, the instance distribution, D_t, from the previous iteration is first normalized (step 1). A hypothesis (classifier), h_t, is generated by training on 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 performance p_t (step 3).

A class conditional weight factor ($w_{t,c}$) is created for each classifier, which is proportional to its classification performance on the entire training data $\mathcal{D}_k$ (including the portion unused during its training) and the number of class c instances on which the classifier was trained (step 4). The weighted majority voting 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 distribution D_t is finally updated according to the performance of H_t (step 7) such that the weights of instances correctly classified by H_t are reduced and those that are misclassified are effectively increased.

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

- Sequence of m_k instances $S=[(x_1,y_1),...,(x_{mk},y_{mk})]$ with labels $y_i \in Y_k = \{1,...,C\}$. Let $\mathcal{C}_{k,c}$ be # of class-c instances in $\mathcal{D}_k$
- Weak learning algorithm **BaseClassifier**.
- Integer T_k, specifying the number of iterations.

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

Initialize $D_1(i) = 1/m_k$, $eT_1 = 0$ $i=1,...,m_k$, set $N_c = \mathcal{C}_{k,c}$

IF $k>1$, **Go to** Step 5, evaluate current ensemble on new data set $\mathcal{D}_k$, update D_t and current # of classifiers $eT_k = \sum_{j=1}^{k-1} T_j$

- Update $eN_c \bullet N_c$ and $N_c = \sum_{j=1}^{k} \mathcal{C}_{j,c}$
- Update $w_{t,c} = w_{t,c} \frac{eN_c}{N_c}$, $t=1,...,eT_k$, $c=1,...,C$

Do for $t = eT_k + 1, eT_k + 2, ..., eT_k + T_k$

1. Set $D_t = D_t \Big/ \sum_{i=1}^{m} D_t(i)$ so that D_t is a distribution.
2. Call **BaseClassifier** providing it with a subset ($\mathbf{d}_k$) of $\mathcal{D}_k$ randomly chosen according to D_t.
3. Obtain a hypothesis $h_t : X \rightarrow Y$, and compute the error
$$\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$$
If $\varepsilon_t > \frac{1}{2}$, discard h_t and go to step 2. Otherwise, compute the normalized performance $p_t = 1 - 2\varepsilon_t$, $0 \le p_t \le 1$.
4. Compute the class specific weight as
$$w_{t,c} = p_t \frac{n_c}{N_c}, \; c = 1,...,C$$
where n_c is the number of class-c instances in $\mathbf{d}_k$
5. Call weighted majority voting to obtain the composite hypothesis $H_t(x_i) = \arg\max_{c\in Y_k} \sum_{t:h_t(x_i)=c} w_{t,c}$
6. Compute the error of the composite hypothesis
$$E_t = \sum_{i:H_t(x_i)\neq y_i} D_t(i)$$
7. Set $B_t = E_t/(1-E_t)$, $0<B_t<1$, and update the instance weights:
$$D_{t+1}(i) = D_t \times \begin{cases} B_t, & \text{if } H_t(x_i) = y_i \\ 1, & \text{otherwise} \end{cases}$$

Call weighted majority voting to obtain the final hypothesis.
$$H_{final}(x_i) = \arg\max_{c\in Y_k} \sum_{t:h_t(x_i)=c} w_{t,c}$$

Fig. 1. Algorithm Learn++.MT2

This distribution update rule, based on the performance of the ensemble, ensures that the algorithm pick and be trained on instances that are difficult, not yet learned, or not previously seen by the ensemble. The algorithm achieves incremental learning, because novel instances introduced by a new dataset are precisely those that are difficult, not yet learned or not yet seen instances.

III. Simulation Results

Learn++.MT2 has been tested on several databases. For brevity, we present results on two benchmark databases obtained from UCI [16], and one real-world application on identifying one of five volatile organic compounds based on chemical sensor data. Base classifiers were all single layer MLPs, normally incapable of learning incrementally, with 12~40 nodes and an error goal of 0.025 ~ 0.05. In each case the data distributions were designed to simulate unbalanced data. Performance of Learn++ on balanced data, can be found in [3,6]. The number of classifiers created in an ensemble can either be predetermined based on prior experience, or determined by cross validation. In this work, a preset number was determined, as explained below, to facilitate the comparison between Learn++ and Learn++.MT2.

A. Wine Recognition Database

The Wine Recognition database features 3 classes (vineyards in Italy) with 13 attributes (alkalinity, acidity, etc.). The database was split into two training and a test set. The data distribution is given in Table I, which is deliberately set to be mildly unbalanced with respect to cardinalities. Each algorithm was allowed to create a total of 20 classifiers (10 on each dataset). This process was repeated 10 times on both algorithms to compare their generalization performance on the test data. The averaged results over 10 runs are shown in Figure 2 and Table II. Figure 2 depicts the generalization performance of each algorithm as new classifiers are added to the ensemble, whereas Table II shows the final generalization performance after each training session (TS).

TABLE I. Data Distribution for Wine Database

Class	Set 1	Set 2	Test
Vineyard 1	*29*	*10*	*20*
Vineyard 2	*41*	*10*	*20*
Vineyard 3	*18*	*10*	*20*

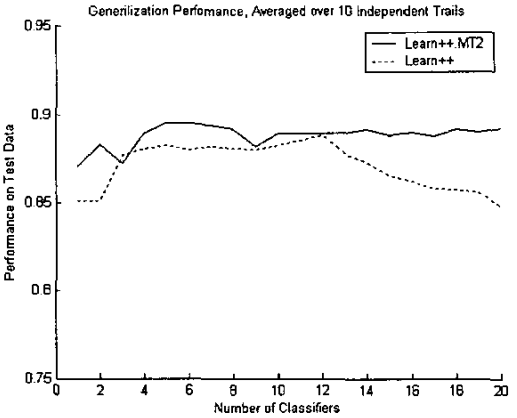

Fig. 2. Simulation Results on the Wine Database

TABLE II. SIMULATION RESULTS ON WINE DATABASE

Algorithm	*TS_1*	*TS_2*	*Std. Dev*
Learn++	*88%*	*84%*	*2.9%*
Learn++.MT2	*88%*	*89%*	*1.6%*

We note that after initial training, the performance of each algorithm is identical, however, Learn++.MT2 outperforms its predecessor after the addition of the second training dataset, with a significant reduction in the standard deviation of the results. We also note the previously mentioned phenomenon of deteriorating performance of Learn++ after 12 classifiers, whereas the performance of new algorithm levels off and stays constant. Therefore, a precise termination point is no longer an issue of significant concern. The differences in the generalization performance between the two algorithms will be addressed below within the context of a more diverse distribution problem.

B. Optical Character Recognition Database

The optical character recognition (OCR) database features 10 classes (digits 0 ~ 9) with 64 attributes. The database was split to create two training and a test dataset, whose distribution can be seen in Table III. Each algorithm was allowed to create 30 classifiers. The data distribution was deliberately made severely unbalanced, specifically designed to test the algorithms' ability to learn under such harsh scenarios. Results from this simulation are shown in Figure 3 and Table IV, which are formatted similar to those in section *A*. All performance percentages are again calculated from an average of 10 independent trials.

TABLE III DATA DISTRIBUTION FOR THE OCR DATABASE

Class	Set 1	Set 2	Test
0	*380*	*10*	*104*
1	*380*	*10*	*121*
2	*380*	*10*	*107*
3	*380*	*10*	*122*
4	*380*	*10*	*118*
5	*380*	*10*	*108*
6	*380*	*10*	*108*
7	*380*	*10*	*116*
8	*380*	*10*	*104*
9	*380*	*10*	*112*

TABLE IV SIMULATION RESULTS ON OCR DATABASE

Algorithm	*TS_1*	*TS_2*	*Std. Dev.*
Learn++	*94%*	*92%*	*0.9%*
Learn++.MT2	*94%*	*95%*	*0.6%*

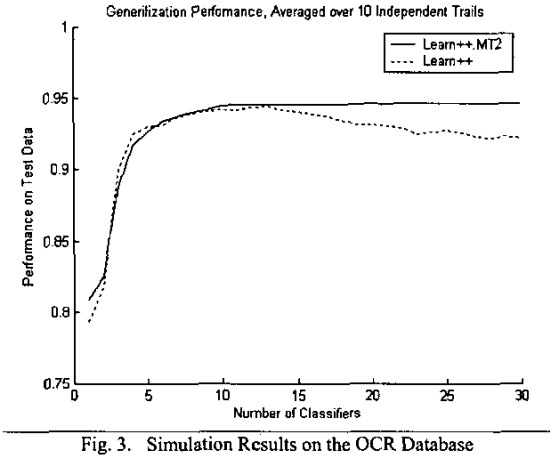

Fig. 3. Simulation Results on the OCR Database

Several interesting observations can be made from these results. First, based on data distribution, one would reasonably agree that the vast majority of discriminatory information is contained in the first dataset, with the second dataset possibly introducing incremental amount of novel information. It is therefore not surprising that both algorithms reached 90% range after the first training. Second, it is also reasonable to argue that classifiers trained on the first dataset should create better representative decision boundaries compared to classifiers trained with the second dataset, and therefore should intuitively be given higher weights. Learn++ which does not take this into consideration gives equal weight (based solely on training performances) to all classifiers, which makes its overall performance more biased towards the second set. Since the second set is not as representative of the overall data, the algorithm's performance declines during this session. Learn++.MT2, however, does take this disproportionality of the datasets into consideration, and is consequently able to retain, even modestly improve by 3%, its knowledge.

C. Volatile Organic Compound Recognition Database

The Volatile Organic Compound (VOC) database consists 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 two training and a test dataset. The distribution of the data is given in Table V, where both dataset cardinalities and the class priors were unbalanced. Both algorithms were incrementally trained with the two subsequent training datasets, and each was allowed to generate up to 30 classifiers. Simulation results, average of 10 independent trials, are shown in Figure 4 and Table VI.

TABLE V. DATA DISTRIBUTION FOR VOC DATABASE

Class	Set 1	Set 2	Test
Ethanol	*30*	*5*	*29*
TCE	*30*	*5*	*29*
Octane	*30*	*5*	*29*
Xylene	*50*	*8*	*22*
Toluene	*80*	*10*	*22*

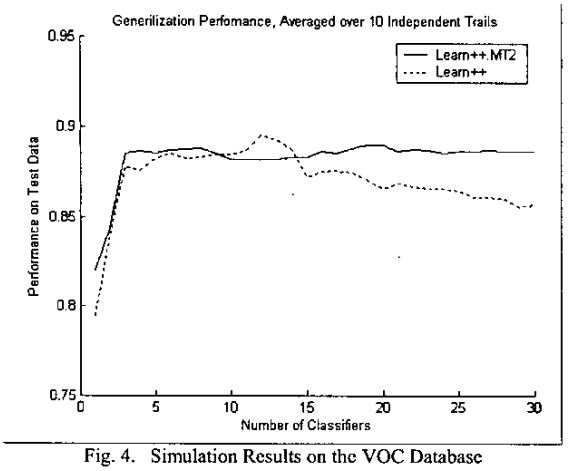

Fig. 4. Simulation Results on the VOC Database

TABLE VI. SIMULATION RESULTS ON VOC DATABASE

Algorithm	*TS_1*	*TS_2*	*Std. Dev.*
Learn++	*89%*	*86%*	*2.1%*
Learn++.MT2	*88%*	*89%*	*1.9%*

The results from Figure 4 and Table VI illustrate a performance characteristic similar to those on the previous databases. As it would be expected based on the data distribution, Learn++.MT2 shows a slight increase in generalization performance during the second training session, while the performance of its predecessor starts declining after 15 classifiers.

D. Effect of Order of Presentation of Data

So far we have presented tests where the initial training session presented more information (instances) followed by a second training session presenting fewer instances. It is conceivable that a practical application could call for the opposite scenario: a limited initial training data followed by large volumes of future training data. This case should be considered not only for application purposes, but also to test the algorithms robustness to the order in which the data are presented. To simulate such a scenario, the OCR database was used. The distribution is similar to Table III except the datasets were introduced in reverse order. In other words, Set 2 was used in the first training session and Set 1 was used in the second training session.

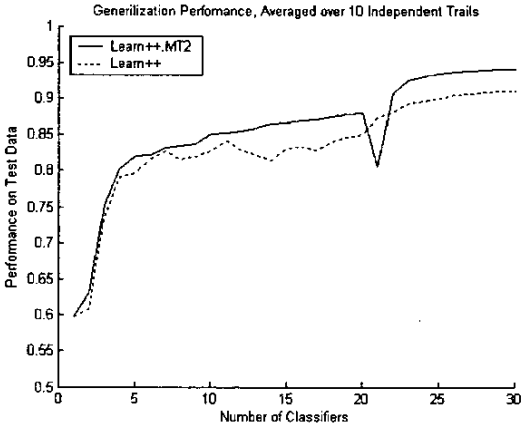

Fig. 5. OCR Reverse Dataset Presentation Results

TABLE VII. OCR REVERSE DATASET PRESENTATION RESULTS

Algorithm	*TS_1*	*TS_2*	*Std. Dev.*
Learn++	*85%*	*91%*	*0.7%*
Learn++.MT2	*88%*	*94%*	*0.6%*

Twenty classifiers were allotted to each algorithm during first training, and an additional ten during the second. Everything else was kept the same. The results shown in Figure 5 and Table VII are, as earlier, averages of 10 independent trials.

These results show that even with the order of presentation reversed, the final generalization performance of both algorithms remains virtually the same, indicating that both algorithms are independent of the order in which datasets are presented. More interestingly, however, the temporary dip in Learn.MT2 performance immediately as the second set is introduced, ironically justifies the approach taken by this algorithm: Since the second dataset introduces a large number of novel instances, the weight of the 21st classifier is considerably larger compared to the now-lowered weights of the previously generated 20 classifiers. However, this one new classifier has only been trained with a subset of the new training data, and on its own has not yet learned the entire data. The ensemble's decision, weighted heavily on this 21st classifier, momentarily lowers the generalization performance. However, the generalization performance immediately recovers, and exceeds that of Learn++, as additional well-trained classifiers are added to the ensemble.

Finally, while we have used MLPs as base classifiers, both algorithms are in fact independent of the type of the base classifier used. The classifier independence of Learn++ was demonstrated and reported in [6].

IV. Discussions and Conclusions

In many applications that call for incremental learning of new information from consecutive datasets, an unbalanced data distribution among the datasets can cause potentially significant performance degradation. This is because, in the absence of any other information, all datasets are assumed to be equally informative, an assumption that may not be valid. Such a scenario would then cause an ensemble based algorithm to give unjustifiably high voting weights to poorly trained classifiers, hence the potential performance degradation.

In this paper, we introduced Learn++.MT2, an ensemble based algorithm specifically designed to handle unbalanced data in incremental learning settings. We have shown that such an imbalance in data distributions, when used to train an ensemble based classifier, does indeed cause overfitting-like behavior where the generalization performance decreases with additional classifiers. When the same datasets were used to train the proposed algorithm, however, this overfitting-like phenomenon was not observed. In none of the unbalanced datasets we have used to train Learn++.MT2, has the performance degraded with the introduction of new data or new classifiers. On the contrary, we have observed typically modest, but sometimes significant, performance improvement:

The novelty of the proposed approach is in its use of class conditional weight factors in assigning voting weights to the classifiers in the ensemble. Specifically, for each classifier, this factor is the ratio of the number of instances from a particular class used for training that classifier, to the number of instances from that class used for training all classifiers thus far in the ensemble. We note that this factor takes both the cardinality of the dataset as well as the class priors into consideration in assigning the voting weights. The actual voting weights are then determined as individual training performances of the classifiers, adjusted by the class conditional weight factors.

The promising preliminary results suggest the proposed approach has the following desirable properties: (i) the algorithm is able to learn novel information content from consecutive datasets, even when such datasets provides severely unbalanced data; (ii) the stability of the algorithm (ability to retain previously acquired knowledge) is improved, without any apparent loss in the plasticity (the ability to acquire new knowledge), which places Learn++.MT2 at a much desirable location on the stability-plasticity spectrum; (iii) the algorithm is independent of the order in which the datasets are presented; or which base classifier is used; (iv) while the proposed approach was developed for and implemented on Learn++, it is fairly general and should benefit any ensemble combination procedure trained on unbalanced data.

Further optimization of the voting weights, evaluation of the proposed approach on multiple datasets introducing additional classes, on other ensemble techniques, as well as on a broader spectrum of applications are currently underway.

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

[3] R. Polikar, L. Udpa, S. Udpa, 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.

[4] 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.

[5] N. Littlestone and M. Warmuth, "Weighted majority algorithm," *Information and Computation*, vol. 108, pp. 212-261, 1994.

[6] R. Polikar, J. Byorick, et al., "Learn++: a classifier independent incremental learning algorithm for supervised Neural Networks," *Proc. of IJCNN 2002*, Honolulu, vol.2, pp. 1742-1747.

[7] 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.

[8] 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.

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

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

[11] T. Ho, J. Hull, S. Srihari, "Decision combination in multiple classifier systems," *IEEE Trans. PAMI*, vol. 16, no. 1, pp. 66-75, 1994.

[12] 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, New York: Springer, 2002.

[13] T. Windeatt and F. Roli (eds), In *Proc. 4th Int. Workshop on Multiple Classifier Systems (MCS2003)*, LNCS, vol. 2709, New York, Springer, 2002.

[14] 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.

[15] T.G. Dietterich, "Ensemble methods in machine learning," *Proc. 1st Int. Wkshop on Multiple Classifier Systems*, LNCS , J. Kitler, F. Roli, ed., vol. 1857, pp.1-15, New York, Springer. 2000.

[16] 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.