Proceedings of the Sixth International Conference on Machine Learning and Cybernetics, Hong Kong, 19-22 August 2007

MULTIPLE CLASSIFIERS BASED INCREMENTAL LEARNING ALGORITHM FOR LEARNING IN NONSTATIONARY ENVIRONMENTS

MICHAEL D. MUHLBAIER, ROBI POLIKAR

Signal Processing and Pattern Recognition Laboratory, Rowan University, Glassboro, NJ 08028 USA
E-MAIL: muhlbaier@ieee.org, polikar@rowan.edu

Abstract:

We describe an incremental learning algorithm designed to learn in challenging non-stationary environments, where the underlying data distribution that governs the classification problem changes at an unknown rate. The algorithm is based on a multiple classifier system that generates a new classifier every time a new dataset becomes available from the changing environment. We consider the particularly challenging form of this problem, where we assume that the previously generated data points are no longer available, even if some of those points may still be relevant in the new environment. The algorithm employs a strategic weighting mechanism to determine the error of each classifier on the current data distribution, and then combines the classifiers using a dynamically weighted majority voting. We describe the implementation details of algorithm, and track its performance as a function of the environment's rate of change. We show that the algorithm is able to track the changing environment, even when the environment changes drastically over a short period of time.

Keywords:

Incremental learning, non-stationary learning, Learn++, ensemble systems, multiple classifier systems.

1. Introduction

Much of the recent history of machine learning research has focused on algorithms than can learn from a set of training data, where the data points are assumed to be drawn from a fixed, yet unknown distribution. A vast majority of the algorithms that are developed over the last several decades, including various forms of neural networks, decision trees, and other statistical learners have made the "unknown but fixed distribution" assumption even in online and incremental learning scenarios. The problem of learning in a non-stationary environment (NSE), where the underlying data distribution changes over time, has received less attention, perhaps due to the difficulty of this problem.

Early work in NSE learning have primarily been on the definition of the problem, and identifying the types of nonstationary environments that can be learned [1-3]. Even defining exactly what constitutes a nonstationary environment is not a trivial matter; after all the change in the environment can be abrupt or gradual, it can be slow or fast, it can be random or systematic, it can be cyclical or not. The common denominator in all of the above mentioned scenarios is that distribution that generates the data changes over time in some manner. Hence the change in the environment is also referred to as *concept drift*.

Learning in non-stationary environments have been receiving an increasing amount of attention, in part due to many practical applications, such as spam detection, that can benefit from an algorithm that can learn in such environments. Several approaches have been proposed for various combinations of above mentioned non-stationary environment scenarios, though many of these algorithms typically employ a procedure for detecting that there is a drift and its magnitude; adjust the learner's parameters to incorporate the change in the environment; and forget what is no longer relevant to the classification problem.

Perhaps one of the earliest examples of algorithms capable of learning in non-stationary environment is the FLORA family of algorithms [1], where a new classifier is trained on a windowed block of training samples, as new data points arrive. Only those instances that fall within the current window are deemed relevant, and hence any information carried by those samples that fall outside of the current data window is automatically forgotten. Other approaches include novelty detection to determine when changes occur [4,5], or treating the concept drift as a prediction problem and use an adaptive neural network that can adjust its parameters according to the environment [6].

The ensemble of classifiers, or multiple classifier systems (MCS) based approaches constitute a new breed of algorithms for learning in nonstationary environments. Such algorithms typically use more than one classifier to track the changing environment. We should note that the algorithms described above also create multiple classifiers, since a new classifier is generated as new data become available. However, these algorithms do not constitute MCS

1-4244-0973-X/07/$25.00 ©2007 IEEE

based approaches, since only one classifier (specifically the one generated last) is used for the classification.

In her recent review [7], Kuncheva puts MCS based approaches into one of three general categories: (i) a fixed ensemble whose combination rules (weights) are changed based on the changing environment [8]; (ii) the new data is used to update the parameters of an online learning algorithm [9]; and/or (iii) add new members to an existing ensemble [10] or replace the least contributing ensemble members with a new one generated based on new data [11].

In this paper, we introduce an alternative MCS based approach that can be described as a hybrid of the above listed strategies. Specifically, we use new data to create new ensemble members, but we also adjust the combination rule based on the errors of the existing classifiers on the new data. The algorithm does not use any of the previously seen data to ensure that the algorithm remains truly incremental, and instead relies on the earlier classifiers to refer to previously learned but still relevant information. Also, the algorithm does not discard any of the previously generated classifiers, in case those classifiers become relevant again should there be a cyclical change in the distribution. The algorithm, named Learn^{++}.NSE is based on our previously introduced – AdaBoost inspired [12] – incremental learning algorithm Learn^{++} [13]. In Learn^{++}, we assumed that additional data was presented in an incremental fashion, however the underlying distribution was assumed fixed.

We first describe the algorithm Learn^{++}.NSE and present promising results of various simulation experiments. We also look at the algorithm's performance as a function of the rate of change. It is reasonable to assume that slower rate of change will make it easier for an online algorithm to track the changes. However, we are interested to see how well the algorithm can still track the changes in the environment as the rate of change increases.

2. Nonstationary Learning Algorithm Learn^{++}.NSE

2.1. Overall description of the algorithm

In this paper, we define the problem of learning in a non-stationary environment as follows: the learning algorithm is presented with a series of training datasets, each of which is drawn from a different snapshot of a distribution that is drifting at an unknown rate. The rate of change may or may not be constant. We further assume that the previously seen data – whether any of it is still relevant or not – is no longer available, or storing previous data is not possible or not allowed. This restriction is inline with most definitions of incremental learning. Any information previously provided by earlier data must then be stored in the parameters of the previously generated classifiers. Unlike most algorithms for nonstationary learning, we also do not use a time window over incoming instances, nor do we use an aged based forgetting. Instead, we generate an ensemble of classifiers from new data, which are then combined by using a strategic weighting mechanism that tracks the classifiers' performances over changing environments to determine appropriate voting weights. Specifically, we assume that each new dataset represents a new snapshot of the then current environment. The amount of change in the environment since the previous dataset – whether fixed or variable, or minor or substantial – is tracked by the performance of the existing ensemble on the current dataset. Learn^{++}.NSE generates a single classifier for each dataset that becomes available, and then combines them through a dynamically weighted majority voting, where the voting weights are computed as weighted averages of classifiers' individual performances over all previous and current environments. The averaging uses a non-linear (sigmoid) function, which gives more weight to performances of the classifiers to the more recent environments.

Consequently, the change in the environment is tracked not only by the addition of new classifiers, but also by the dynamic adjustment of the voting weights of the existing classifiers. No classifier is ever discarded, since previously generated classifiers can possibly become relevant and informative, if a cyclical environment returns to the original distribution, or if only some of the class conditional distributions experience concept drift. In such cases, Learn^{++}.NSE recognizes the relevance of earlier classifiers, and awards them with higher weights. The algorithm is described in detail in the next section, whose pseudocode is provided in Figure 1.

2.2. Detailed description of the algorithm

The inputs to the algorithm are the BaseClassifier model that generates individual classifiers, and the training data at each time snapshot. The training dataset $\mathbb{D}^t$ at time t provides us with m^t data points that serves as a snapshot of the then current environment. We assume that the distribution that generated the dataset $\mathbb{D}^t$ has changed in some manner and rate unknown the us, since the previous dataset $\mathbb{D}^{t-1}$ was provided at time t-1.

Learn^{++}.NSE generates one classifier h_t for each such new dataset that becomes available, which are then combined with all previous classifiers to create the composite hypothesis H_t. The decision of H_t then serves as the ensemble decision. More specifically, the algorithm first evaluates the classification accuracy of the currently available composite hypothesis H^{t-1} on the newly available data $\mathbb{D}^t$. H^{t-1},

obtained by the weighted majority voting of all classifiers generated during the previous t-1 iterations, represents the existing ensemble's knowledge of the current environment. The error of the composite hypothesis, E^t, is computed as the ratio of correctly identified instances of the new dataset. We require that this error be less than a threshold, typically ½ that allows a convenient normalization, to ensure that H_t has a meaningful classification capacity.

Input: For each new dataset D^t $t=1,2,...$

- Training data $\{x_i, y_i | x_i \in R, \quad y_i \in Y = \{1,\ldots,c\}$ $i=1,\ldots,m^t$
- Supervised learning algorithm **BaseClassifier**.

Do for $t=1,2,...$

If $t=1, D^1(i) = w^1(i) = 1/m^1, \forall i$, skip to step 3. (1)

1. Compute error of existing ensemble on new data

$$E^t = \sum_{i=1}^{m^t} \left(1/m^t\right) \cdot \left[\left[H^{t-1}(x_i) \neq y_i\right]\right] \quad (2)$$

Normalize error $B^t = E^t / \left(1 - E^t\right)$ (3)

2. Update instance weights

$$w_i^t = \frac{1}{m^t} \times \begin{cases} B^t, & H^{t-1}(x_i) = y_i \\ 1, & otherwise \end{cases} \quad (4)$$

Set $D^t = w^t \Big/ \sum_{i=1}^{m^t} w_i^t$ so that D^k is a distribution. (5)

3. Call **BaseClassifier** with D^t, obtain $h_t : X \rightarrow Y$

4. Evaluate all existing classifiers on new dataset D^t

$$\varepsilon_k^t = \sum_{i=1}^{m^t} D^t(i) \cdot \left[\left[h_k(x_i) \neq y_i\right]\right], k = 1,...,t \quad (6)$$

If $\varepsilon_{k=t}^t > 1/2$, generate a new h_t. If $\varepsilon_{k<t}^t > 1/2$,

set $\varepsilon_k^t = 1/2$, $\beta_k^t = \varepsilon_k^t / \left(1 - \varepsilon_k^t\right), k = 1,...,t$ (7)

5. Compute a weighted sum of all normalized errors for k^{th} classifier h_k

$$\omega_k^t = 1 \Big/ \left(1 + e^{-a(t-k-b)}\right), \omega_k^t = \omega_k^t \Big/ \sum_{j=0}^{t-k} \omega_k^{t-j} \quad (8)$$

$$\bar{\beta}_k^t = \sum_{j=0}^{t-k} \omega_k^{t-j} \beta_k^{t-j}, \quad \text{for } k = 1,...,t \quad (9)$$

6. Calculate classifier voting weights

$$W_k^t = \log\left(1/\bar{\beta}_k^t\right), \quad \text{for } k = 1,...,t \quad (10)$$

7. Obtain the composite hypothesis as the current final hypothesis

$$H^t(x_i) = \arg\max_c \sum_k W_k^t \cdot \left[\left[h_k(x_i) = c\right]\right] \quad (11)$$

Figure 1. Learn^{++}.NSE algorithm

For this selection of the threshold, we normalize the error so that the normalized error B^t remains between 0 and 1 (step 1 inside the Do loop). We then update a set of weights for each instance, that is normally initialized to be uniform, such that the weights of the instances misclassified by the ensemble are reduced by a factor of B^t . The weights are then normalized to obtain a distribution D^t (step 2).

The algorithm then calls the BaseClassifier and asks it to create the t^{th} classifier h_t using data drawn from the current training dataset D^t (step 3). All classifiers generated thus far h_k, k=1,...,t are then evaluated on the current dataset, by computing their weighted error (Equation 6, step 4). Note that at current time step t, we now have t error measures, one for each classifier generated thus far. Hence the error term ε_k^t represents the error of the k^{th} classifier h_k at the t^{th} time step.

The errors are again assumed to be less than ½. Here, we make a distinction: if the error of the most recent classifier on its own training data is greater than ½, we discard that classifier and generate a new one. After all, if it cannot perform at least 50% on the training data it has just seen, than this last classifier is of very little use. For any of the other (older) classifiers, if its error is greater than ½, it is set to ½. This effectively sets the normalized error of that classifier at that time step to 1, which in turn removes all of its voting power later during the weighted majority voting. Note that unlike the current classifier, previously generated classifiers are not discarded when their error exceeds ½. This is because, it is not unreasonable for a classifier to perform poorly on a future dataset, if the environment has changed drastically since it was created. Furthermore, this classifier can very well be useful in the future, should the environment returns to the condition it was in when the classifier was generated. If the future distributions are different enough that the previous classifiers are not useful, they are made dormant by setting their error rate to ½. If, on the other hand, these classifiers become relavant again in the future, then such relevance is reflected by a lower (than ½) error rate they obtain on the then current environment.

In order to combine the classifiers, however, we need one weight for each, even though the algorithm maintains a set of t such error measures ε_k^t for each classifier k=1,…,t. The error measures are therefore combined through a weighted averaging (step 5). The weighting is done through a nonlinear sigmoid function (Equation 8), which gives a large weight to errors on most recent environments, and less to errors on older environments. Note that this process does *not* give less weight to old classifiers: it gives less weight to their *error on old environments*. Therefore, a classifier generated long time ago can in fact receive a large voting weight, if its error on the *recent* environments becomes low.

The errors so weighted are combined to obtain a

weighted average error (Equation 9). The logarithm of the inverse of this weighted error average then constitutes the final voting weight W_k^t for classifier k at time instant t (Step 6, Equation 10). Finally all classifiers are combined through weighted majority voting (Step 7, Equation 11).

3. Simulation Experiments and Results

We describe two experiments that are designed to illustrate the performance and behavior of the algorithm in two different and challenging non-stationary learning scenarios. In both experiments, we use data that is originally drawn from Gaussian distribution, so that we can compare the algorithm's performance to that of the optimal Bayes classifier, and to the performance of the single naïve Bayes classifier that is always trained on the most recent dataset.

In the first case we evaluate the ability of the algorithm to track the changing environments with respect to the rate of change of the environment for a two-dimensional, three class problem. The environment and its changing path are illustrated in Figure 2. The actual parametric equations used to determine the changing environment are shown in Table 1. Note that the class means μ_x and μ_y change in all three classes but the variance only changes in class 3. The environment changes from t=0 to t=1 in T time steps, and at each step 25 instances are drawn from the environment to train a Naïve Bayes classifier. This classifier is then added to the Learn++.NSE ensemble, to be combined with the previous classifiers as described above. This process is repeated for an array of T values (T = 10, 20, 30, 40, 50, 65, 80, and 100). Note that a small value of T indicates a fast changing environment, and a large value of T indicates a slow changing environment. Figure 3 shows the performance of Learn++.NSE, Naïve Bayes, and Bayes at each time step for each experiment using a different T value. All performance plots are averages of 100 independent trials.

Several interesting observations can be made from Figure 3. First and foremost, we note that the ensemble performance is always as good (T=10) or better than that of the single classifier (all other values of T). Note that T=10 is the fastest changing environment among all trials, with only 10 time steps allowed for the algorithm to track the entire change. Even then, the ensemble performance at worst is the same as that of a single classifier that has seen the most recent dataset. Second, as expected, the ensemble performance improves as the rate of change decreases. This makes sense, since a slower changing environment is easier to track. As T increases, the previously generated classifiers in the ensemble become more relevant to the current environment, thus increasing the performance. Third, as T continues to increase, the performance gain of the algorithm levels off, as it is able to completely adapt to the changes in the environment. Table 2 shows the average error rate, over all T time steps, normalized to the error rate of the Bayes classifier. Note that the relative error of Naïve Bayes remains constant (with respect to optimal Bayes error), where as the ensemble performance improves as T increases.

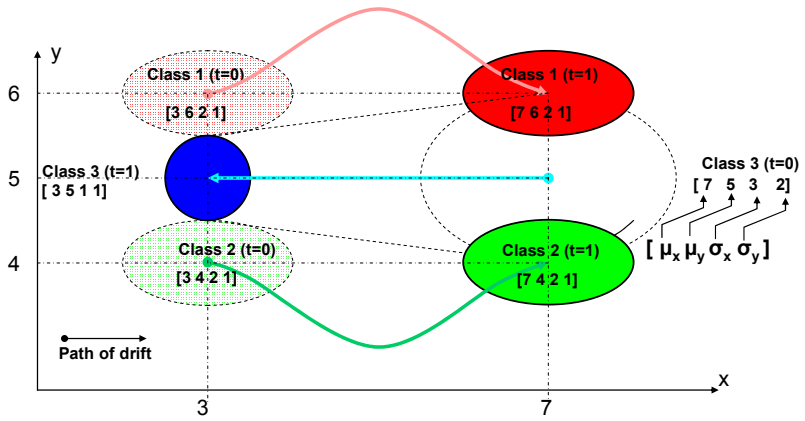

Figure 2. First simulation experiment on a three class data, where both class means and variances change

Table 1. Parametric expressions controlling the change of class mean and standard deviations for Gaussian data

Data	μ_x	μ_y	σ_x	σ_y
Class 1	3+4t	3.5+0.5cos(2πt)	2	1
Class 2	3+4t	6.5-0.5cos(2πt)	2	1
Class 3	7-4t	5	3-2t	2-t

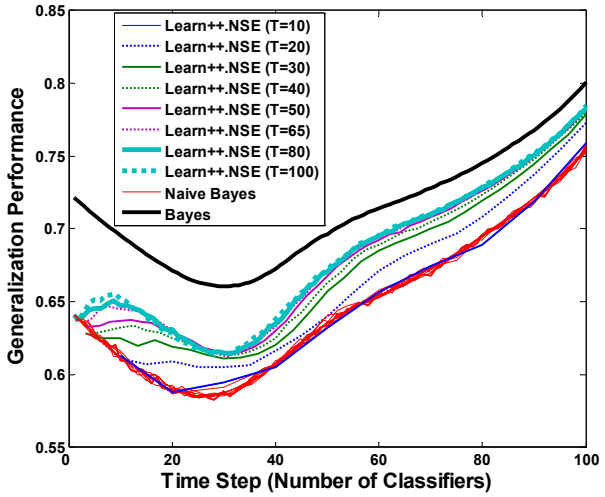

Figure 3. Learn^{++}.NSE performance against the number of classifiers for different rates of change.

Finally, we also observe that the algorithm performance approaches to that of optimal Bayes performance as new members are added to the ensemble, regardless the value of T. Therefore, the algorithm does not suffer from catastrophic forgetting. This is attributed to the boosting based structure of the algorithm. The resistance of boosting based approaches to overfitting is a previously observed phenomenon, which has been explained by the margin theory [14].

Table 2. Average error, relative to the Bayes classifier

	Learn++.NSE	Naïve Bayes
T = 10	6.2%	6.3%
T = 20	5.1%	6.4%
T = 30	4.2%	6.4%
T = 40	3.7%	6.4%
T = 50	3.4%	6.3%
T = 65	3.2%	6.4%
T = 80	3.1%	6.4%
T = 100	3.1%	6.4%

Note that in this first experiment the environment changes gradually, even for small values of T which increased the rate of change. In our second experiment, we introduced several sharp, abrupt and unexpected changes to the distribution in order to observe the behavior of the algorithm in such scenarios. Figure 4 illustrates the changing environments of this challenging scenario. The experiment consists of four classes, each with a Gaussian distribution. The environment periodically changes such that each class' distribution rotates in a counter

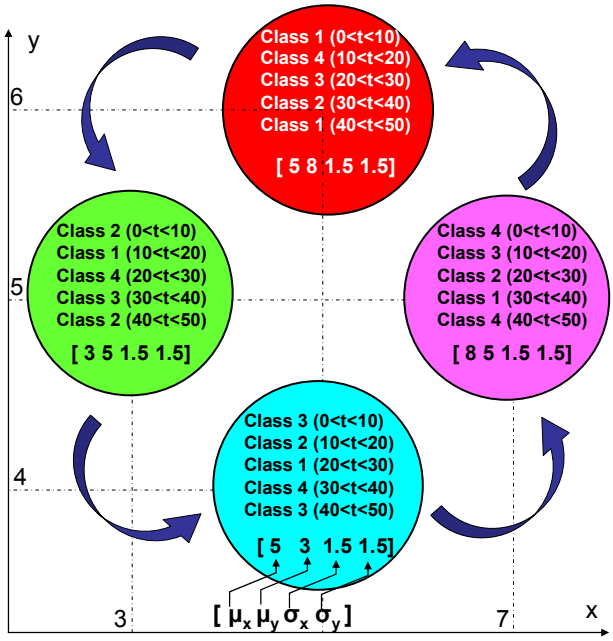

Figure 4. Rotating classes problem

clockwise manner, moving to where another class was previously located. Hence, any classifier trained before the abrupt change would be incapable of correctly classifying data from the current environment.

The performances of the Bayes, single classifier Naïve Bayes and Learn^{++}.NSE are shown in Figure 5. We now observe that Learn^{++}.NSE initially takes advantage of the boosting characteristics inherent in its structure and reduces its error as classifiers are added to the ensemble (up until classifier 10).

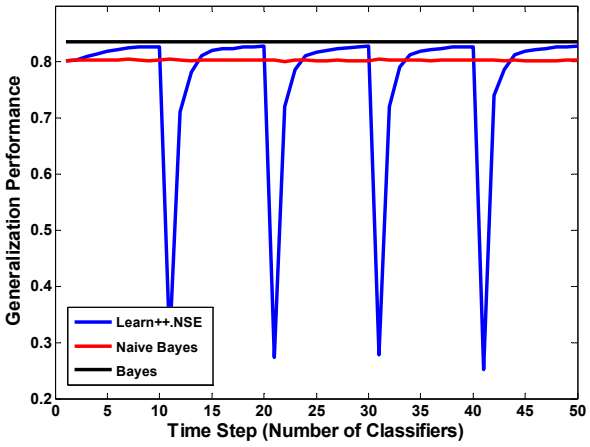

Figure 5. Learn^{++}.NSE performance in abruptly changing environments

However, when the environment changes abruptly and drastically, the performance of the algorithm immediately plunges. This is not unexpected, since only one classifier (the one generated the last) in the ensemble can correctly recognize the new environment at the time of the abrupt change. However, the performance of the ensemble based Learn^{++}.NSE rapidly increases as new classifiers are added, out performs the single Naïve Bayes classifier, and in fact approaches to the optimal Bayes classifier.

4. Conclusions

We described an MCS based algorithm for learning in nonstationary environments. The algorithm creates a single classifier for each dataset that becomes available, and keeps a record of the performance of each classifier on all environments throughout the training. The classifiers are then combined through a dynamically weighted majority voting, where the voting weights are determined by each classifier's performance on the current environment, weighted along with the performances on previous environments. All classifiers are retained, which allows the previously generated classifiers to make significant contributions to the ensemble decision, if such classifiers provide relevant information for the current environment.

On two experiments, we showed that the algorithm can track the changing environments regardless of the rate of change, though the performance markedly increases for slower rates of change. We further showed that the algorithm can also track abrupt and rapid changes, as soon as a reasonable ensemble size (greater than 1) is obtained for each new dataset.

This algorithm is currently in its infancy stages of its development; however, initial results are promising and warrant further analysis of this approach.

Acknowledgements

This material is based on work supported by the National Science Foundation under Grant No: ECS 0239090.

References

[1] Widmer, G. and Kubat, M.; "Learning in the presence of concept drift and hidden contexts," *Machine Learning* vol. 23, no. 1, pp. 69-101, 1996.

[2] Schlimmer, J. C. and Granger, R. H.; "Incremental learning from noisy data," *Machine Learning*, vol. 1, no. 3, pp. 317-354, 1986.

[3] Helmbold, D. P. and Long, P. M.; "Tracking drifting concepts by minimizing disagreements," *Machine Learning*, vol. 14, no. 1, pp. 27-45, 1994.

[4] Gama, J., Medas, P., Castillo, G., and Rodrigues, P.; "Learning with drift detection," *SBIA 2004, Lecture Notes in Comp. Science*, vol.3171, pp.286-295, 2004.

[5] Cohen, L., Avrahami-Bakish, G., Last, M., Kandel, A., and Kipersztok, O.; "Real-time data mining of non-stationary data streams from sensor networks," *Information Fusion*, in press, (2007).

[6] Rutkowski, L.; "Adaptive probabilistic neural networks for pattern classification in time-varying environment," *IEEE Trans. on Neural Networks* vol. 15 811-827, no. 4, 2004.

[7] Kuncheva, L. I.; "Classifier Ensembles for Changing Environments," *Multiple Classifier Systems (MCS 2004), Lecture Notes in Computer Science* vol. 3077, pp. 1-15, 2004.

[8] Blum, A.; "Empirical support for Winnow and weighted-majority algorithms: Results on a calendar scheduling domain," *Machine Learning*, vol.26, no. 1, pp. 5-23, 1997.

[9] Oza, N.; Online Ensemble Learning, Ph.D. Dissertation, (2001) University of California, Berkeley.

[10] Kyosuke, N., Koichiro, Y., and Takashi, O.; "ACE: Adaptive classifiers-ensemble system for concept-drifting environments," *Multiple Classifier Systems (MCS 2005), Lecture Notes in Computer Science*, vol. 3541, pp. 176-185, 2005.

[11] Street, W. N. and Kim, Y.; "A streaming ensemble algorithm (SEA) for large-scale classification," *7th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-01),* pp. 377-382, 2001.

[12]Y. Freund and R. E. Schapire, "Decision-theoretic generalization of on-line learning and an application to boosting," *J. of Comp. and System Sci.*, vol. 55, no. 1, pp. 119-139, 1997.

[13] Polikar, R., Upda, L., Upda, S. S., and Honavar, V.; "Learn++: an incremental learning algorithm for supervised neural networks," *IEEE Transactions on Systems, Man and Cybernetics*, vol.31, no. 4, pp. 497-508, 2001.

[14] Schapire R., Freund Y., Bartlett B., and Lee W., "Boosting the margin: new explanation for the effectiveness of voting methods," *The Annals of Statistics*, vol. 26, no. 5, pp. 1651-1686, 1998.