Learning Concept Drift in Nonstationary Environments Using an Ensemble of Classifiers Based Approach

Matthew Karnick, Metin Ahiskali, Michael D. Muhlbaier and Robi Polikar*

Abstract –**We describe an ensemble of classifiers based approach for incrementally learning from new data drawn from a distribution that changes in time, i.e., data obtained from a nonstationary environment. Specifically, we generate a new classifier using each additional dataset that becomes available from the changing environment. The classifiers are combined by a modified weighted majority voting, where the weights are dynamically updated based on the classifiers' current and past performances, as well as their age. This mechanism allows the algorithm to track the changing environment by weighting the most recent and relevant classifiers higher. However, it also utilizes old classifiers by assigning them appropriate voting weights should a cyclical environment renders them relevant again. The algorithm learns incrementally, i.e., it does not need access to previously used data. The algorithm is also independent of a specific classifier model, and can be used with any classifier that fits the characteristics of the underlying problem. We describe the algorithm, and compare its performance using several classifier models, and on different environments as a function of time for several values of rate-of-change.**

I. INTRODUCTION

A particularly challenging problem in computational intelligence is the ability of a classifier to learn from incrementally updated data drawn from a nonstationary environment, where the underlying data distribution changes in time. Solving this problem requires an algorithm that can track such changes and update the decision boundaries accordingly. Further complication arises if previously learned information is still partially relevant, yet the data it generated is no longer available. Such a scenario requires an *incremental learning* algorithm that strikes a very fine balance between stability and plasticity. The stability allows the algorithm to retain any information that is still relevant, whereas the stability allows the algorithm to acquire new knowledge. These two requirements are often conflicting in nature, and therefore known as the stability-plasticity dilemma [1].

Perhaps due to lack of a standard and formal definition of the problem, or perhaps due to the inherent difficulty of the problem, learning in nonstationary environments (NSE) has traditionally received relatively little attention from the computational intelligence and machine learning communities. A vast majority of the machine learning research has focused, instead, on developing and optimizing a variety of algorithms that make one fundamental assumption: the data are drawn from a fixed but unknown distribution.

There is, however, an increasing level of interest in nonstationary learning problems, which may in part be due to many challenging applications for which traditional algorithms have been shown to be suboptimal. Such applications include spam or fraud detection, analysis of long term financial, epidemiological, climate or demographic data. In all such applications, the distribution that generates the data is known to change – or drift – over time, a phenomenon also known as *concept drift*.

Consider an environment from which we obtain a series of data in batches over a period of time. We say that the environment is nonstationary – and hence there is concept drift – if there is any alteration in the underlying data distribution between any two consecutive time steps. The drift in the concepts (i.e., classes, or class boundaries for classification problems) can be gradual or abrupt, deterministic or random, contracting or expanding, and even cyclical. It is clear, however, some restrictions must be imposed on the concept change for the concept to be learnable – after all, no classifier can learn a function that changes entirely at random. Earliest work in NSE learning has therefore explored different types of nonstationary learning, and identifying the types of nonstationary environments that can in fact be learned [2-5]. For example, it was shown that concept drift can be learned if either the amount [2] or the rate [3] of drift is restricted. Then, a typical concept drift algorithm needs to implement one or more of the following procedures: i) detect that there is a drift; ii) detect its magnitude; iii) adjust its parameters to learn the (drifted) new concept; and iv) forget what is no longer relevant.

A commonly used approach is to use a sliding window on incoming data, and train a new classifier with the latest data that fall within the window. In this case, it is assumed that the rate of change is slow enough that the data within the window does not exhibit any drift. Such an assumption is of course not usually met in practice. A variation of this approach is then to use a variable length window, as in the FLORA family of algorithms [5;6]. The approach has a built-in forgetting mechanism: 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. A related approach is to determine which previous instances are most similar to the current batch of instances, and train the current classifier on all such instances [7], which may also be weighted based on their estimated relevance. This approach, also called instance selection / weighting, is also similar to that of partial memory learning [8], where only those instances that are believed to be the most relevant are used for training.

Other approaches include novelty detection to determine when concept drift occurs [9-11], or treating the

Manuscript received December 15, 2007. This work was supported by the National Science Foundation under Grant No: ECS 0239090.

Authors are with the Electrical and Computer Engineering Department, Rowan University, Glassboro, NJ 08028 USA. *Corresponding author: R. Polikar (e-mail: polikar@rowan.edu).

978-1-4244-1821-3/08/$25.00©2008 IEEE

concept drift as a prediction problem and use an adaptive neural network that can adjust its parameters according to the environment [12]. An overview of these approaches can be found in [13].

A relatively new group of algorithms used for nonstationary learning is the ensemble of classifiers, or multiple classifier systems (MCS) based approaches. These algorithms use more than one classifier to track the changing environment. The aforementioned algorithms, such as FLORA, also create multiple classifiers, since a new classifier is generated as new data become available. However, these algorithms do not constitute MCS based approaches, since only one classifier (specifically the one generated last) is used for classification.

In her recent review [14], 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 (dynamic combiners), as in Winnow [15]; (ii) the new data is used to update the parameters or members of an online learning algorithm, as in Oza's online boosting [16]; and/or (iii) alter the structure of the ensemble by adding new members to an existing ensemble, such as Nishida's adaptive classifier ensemble (ACE) which uses a combination of online and batch classifiers [17], or replace the least contributing ensemble members with a new one generated based on new data, such as Street's streaming ensemble algorithm (SEA) and Kolter's dynamic weighted majority (DWM) [18;19].

Recently, we introduced an alternative MCS based approach, that can perhaps best described as a hybrid of the above listed strategies [20;21]. 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. Furthermore, 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. We call this algorithm Learn^{++}.NSE, which is based on our previously introduced – and AdaBoost inspired [22] – incremental learning algorithm Learn^{++} [23].

In this paper, we describe the algorithm and evaluate its performance on several scenarios, where we alter the rate-of-change of the environment, as well as the base classifier model used to train the ensembles. We show that the algorithm closely tracks the performance of the optimal Bayes classifier, and routinely beats a single classifier that is trained on the latest training dataset.

II. LEARN^{++}.NSE

A. An overview of the algorithm

We begin with the specific interpretation of concept drift as it is used in this paper. The learning algorithm is provided with a series of training datasets $\{\boldsymbol{x_i^t} \in X; y_i^t \in Y\}, i = 1, \cdots, m^t$, where t is an index on the changing environments; hence $\boldsymbol{x}_i^t$ is the i^{th} instance obtained from the t^{th} dataset (environment), drawn from an unknown distribution $P^t(\boldsymbol{x}, y)$, which is the current snapshot of a possibly drifting distribution at time t. At time $t+1$, we obtain a new training dataset drawn from $P^{t+1}(\boldsymbol{x}, y)$. At each time step there may or may not have been a change in the environment, and if there were, the rate of this change is not known, nor assumed constant. Furthermore, we presume previously seen datasets – whether any of them is still relevant or not – are no longer available, or storing previous data is not possible or not allowed. Hence, we require the algorithm to work in an incremental fashion. Note that most instance selection or instance weighting approaches do not make this restriction. Any information previously provided by earlier data must necessarily be stored in the parameters of the previously generated classifiers.

Based on this description of the nonstationary environment, the proposed algorithm Learn^{++}.NSE generates a new classifier every time a new dataset becomes available. Note that the algorithm does not use a sliding window to choose the instances, nor does it fix the number of instances used to train each classifier. Perhaps more importantly, unlike most other algorithms, Learn^{++}.NSE does not permanently discard any of the older classifiers based on their age or even on their current performance.

Instead, Learn^{++}.NSE employs a dynamically updated weighted majority voting for combining the classifiers, strategically controls which classifiers are allowed to vote in the final decision, and by how much. While the previous datasets are not stored or used, the performance of the classifiers on previous datasets are stored. The voting weights are then determined by a weighted average of the classifiers' individual performances on current and past datasets, where current environments are more heavily weighted using a sigmoid-type weighting function. Hence, rather than removing previously generated classifiers, their voting weights are reduced or even (temporarily) nullified if their performance on the current data fall below a certain threshold. This approach then allows the algorithm to use older classifiers, which may become useful again if a cyclical environment returns to its earlier states, 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.

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. We describe the algorithm in detail in the next section, whose pseudocode is given in Figure 1.

B. Algorithm description

The algorithm has two inputs: a supervised classification algorithm, BaseClassifier, to train individual classifiers, and the training data $\mathfrak{D}^t$ drawn from the current distribution $P^t(\boldsymbol{x}, y)$ at time t. The training dataset $\mathfrak{D}^t$ of cardinality m^t serves as a snapshot of the current environment. As mentioned above, we assume that the distribution $P^t(\boldsymbol{x}, y)$ has changed, in some manner and rate unknown us, since the previous distribution $P^{t-1}(\boldsymbol{x}, y)$.

Learn^{++}.NSE maintains a distribution $D^t(i)$ over the training data instances $\boldsymbol{x_i}$. $D^1(i)$ is initialized to be uniform giving equal probability to each instance being drawn from the first dataset. When new data arrives at time t, the algorithm generates one new classifier h_t, which is then combined with all previous classifiers to create the composite hypothesis H_t. The decision of H_t serves as the ensemble decision. Before a new h_t is trained, however, Learn^{++}.NSE first evaluates the classification performance

of the currently available composite hypothesis H^{t-1} on the new dataset $\mathfrak{D}^t$. This represents the existing ensemble's knowledge of the current environment. The error of the composite hypothesis, E^t, is computed on the new dataset, which is then used to update the distribution weights (steps 1 and 2 within the Do loop in Figure 1). The distribution update rule decreases the probability of the correctly classified instances being selected into the next training set, since that part of the feature space covered by those instances are already known by the ensemble. The new training dataset for the next classifier is then drawn from this distribution, which then focuses on those instances that are not yet learned by the current ensemble.

Input: For each dataset $\mathfrak{D}^t$ $t = 1,2, \ldots$
Training data $\{\boldsymbol{x_i^t} \in X; y_i^t \in Y = \{1, \ldots, c\}\}, i = 1, \cdots, m^t$.
Supervised learning algorithm **BaseClassifier**
Do for $t = 1,2, \ldots$

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

1. Compute error of the existing ensemble on new data
$E^t = \sum_{i=1}^{m^t} (1/m^t) \cdot [\![H^{t-1}(x_i) \neq y_i]\!]$ (2)
2. Update and normalize instance weights
$w_i^t = \frac{1}{m^t} \cdot \begin{cases} E^t, H^{t-1}(x_i) = y_i \\ 1, otherwise \end{cases}$ (3)
Set $D^t = w^t / \sum_{i=1}^{m^t} w_i^t \implies D^t$ is a distribution (4)
3. Call **BaseClassifier** with $\mathfrak{D}^t$, obtain $h_t: X \to Y$
4. Evaluate all existing classifiers on new data $\mathfrak{D}^t$
$\varepsilon_k^t = \sum_{i=1}^{m^t} D^t(i) \cdot [\![h_k(x_i) \neq y_i]\!]$ for $k = 1, \ldots, t$ (5)
If $\varepsilon_{k=t}^t > 1/2$, generate a new h_t.
If $\varepsilon_{k<t}^t > 1/2$, set $\varepsilon_k^t = 1/2$,
$\beta_k^t = \varepsilon_k^t / (1 - \varepsilon_k^t)$, for $k = 1, \ldots, t$ (6)
5. Compute the weighted average of all normalized errors for k^{th} classifier h_k: For $a, b \in \mathbb{R}$
$\omega_k^t = 1/\left(1 + e^{-a(t-k-b)}\right)$, $\omega_k^t = \omega_k^t / \sum_{j=0}^{t-k} \omega_k^{t-j}$ (7)
$\bar{\beta}_k^t = \sum_{j=0}^{t-k} \omega_k^{t-j} \beta_k^{t-j}$, for $k = 1, \ldots, t$ (8)
6. Calculate classifier voting weights
$W_k^t = \log(1/\bar{\beta}_k^t)$, $for\ k = 1, \ldots, t$ (9)
7. Obtain the final hypothesis
$H^t(x_i) = \arg\max_c \sum_k W_k^t \cdot [\![h_k(x_i) = c]\!]$ (10)

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

Once the distribution is updated, the algorithm calls the BaseClassifier and asks it to create the t^{th} classifier h_t using data drawn from the current training dataset $\mathfrak{D}^t$ (step 3). All classifiers generated thus far h_k, k=1,...,t are evaluated on the current dataset, by computing ε_k^t, the error of the k^{th} classifier h_k at the t^{th} time step (Equation 5, step 4). At current time step t, we now have t error measures, one for each classifier generated thus far.

These error values are used as follows: 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, then this last classifier is of very little use. We do not hold the old classifiers to the same standard however: if their 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, and removes all voting power of that classifier. When the errors are normalized (Equation 6), those in the [0 ½] interval are mapped to [0 1] range. A normalized error of β_k^t =1, carries a voting weight of 0 in the weighted majority voting (Equation 9). Note that unlike the current classifier, previously generated classifiers are not discarded when their error exceeds ½. The reason for this double standard is that it is not unreasonable for a classifier to perform poorly on a future dataset, if the environment has changed drastically since it was created. This classifier can still be useful in the future, however, should a cyclical environment returns to an earlier state 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 will be reflected by a lower than ½ error rate they obtain on the then current environment, and will therefore receive nonzero voting weights.

The errors are used to determine the voting weight of each classifier using the following process. We first obtain a weighted average of all t error measures ε_k^t for each classifier h_k. A nonlinear sigmoid function is used for the weighting (Equation 7), such that a large weight is given to errors on most recent environments, and a smaller weight is given to errors on older environments. Note that the weights ω_k^t used for averaging the error rates are not the voting weights (which are denoted as W_k^t). The error averaging weights ω_k^t are used to weigh normalized errors β_k^t to obtain $\bar{\beta}_k^t$ (Equation 8); the logarithm of $1/\bar{\beta}_k^t$ is the final voting weight W_k^t of classifier h_k at time t.

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 still receive a large voting weight, if its error on the *recent* environments is small. Finally, all classifiers are combined through weighted majority voting (Step 7, Equation 10).

III. Simulation Experiments and Results

A. Experimental Setup

We designed two experiments to demonstrate several traits and properties of the proposed algorithm. In both experiments, we created a nonstationary environment where the data distribution – chosen to be Gaussian – changes continuously over time. We chose Gaussian distribution so that the performance of the algorithm could be compared to that of an optimal Bayes classifier. In both cases, we have varied the rate of change in order to evaluate the ensemble's adaptability to environments that change at different rates. To do so, we assume that the environment is at its initial state at t = 0, and at each consecutive time instance t we receive data from an environment that has drifted by some amount, until some arbitrary time instance t = 1, where we terminate the experiment. We use a parameter T to control how many time steps are observed between t = 0 (start) and t = 1 (end). A larger T value indicates that we observed many intermediate updates from the environment between the start and end, indicating a smoother and slower drift at each step. Conversely, smaller values of T represent abrupt changes, as we observe fewer

datasets between the start and end points. At each time step t, $m' = 20$ instances are drawn from the current distribution that make up $\mathfrak{D}^t$ which are used to train the current base classifier. We have repeated the experiments for several values of $T = \{10, 20, 40, 60, 80, 100\}$.

We further repeated all experiments using three different base classifiers to evaluate whether the algorithm can work with different base classifiers. We chose the multilayer perceptron (MLP), support vector machine (SVM) and naïve Bayes to be used as base classifiers. Note that the MLP and SVM – in their original formulation – are strictly batch-learning algorithms that are not capable of online learning, whereas naïve Bayes can be used as an online learning algorithm.

Finally, for each experiment we also compared the performance of the ensemble created by Learn^{++}.NSE to that of a single classifier trained on the latest dataset. Note that this is not a trivial comparison: considering that the test data always come from the current (latest) environment, a single classifier trained on the data generated by that last environment is most likely to do better than all other classifiers trained on previous environments. For an ensemble-based system to achieve an improvement over such a single classifier, the ensemble members must provide relevant information – about the current environment – even though all but one of the ensemble members were trained on data obtained from a – now partially irrelevant - past environment.

B. *Simulation Results – Experiment 1*

In our first experiment, we generated a four-class dataset drawn from a two-dimensional Gaussian distribution. Figure 2 shows the probability distributions at four instances $t = 0$, $t = 1/3$; $t = 2/3$; and $t = 1$, whereas Table 1 shows the parametric equations used to calculate the path of the drift, where each row represents how the distribution of one of the four classes was varied. The parametric equations themselves were also changed at $t = 1/3$ and $t = 2/3$. For example, during the first phase of $t = 0$ to $t = 1/3$, all class means remained the same, but the x-direction (horizontal) standard deviations of class 3 (C_3) and class 4 (C_4) distributions were shrunk from $\boldsymbol{\sigma}_x = 3$ to $\boldsymbol{\sigma}_x = 1$ through the parametric expression $3\text{-}6t$ and C_1's standard deviation increased from $\boldsymbol{\sigma}_x = 1$ to $\boldsymbol{\sigma}_x = 2$ through the parametric expression $1+6t$. As another example, during $t = 2/3$ to $t = 1$, C_1 means moved from $\boldsymbol{\mu} = [2,5]$ to $\boldsymbol{\mu} = [5,2]$, whereas its standard deviations moved from $\boldsymbol{\sigma} = [1,6]$ to $\boldsymbol{\sigma} = [1,2]$ (all cross covariances were zero).

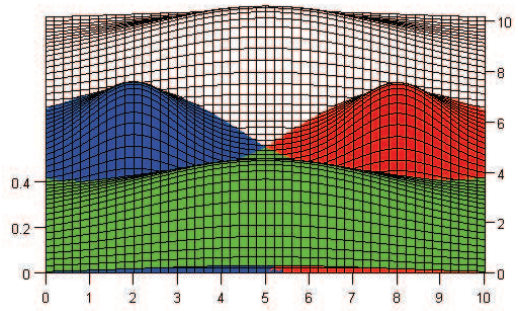

Fig. 2.a: Snapshot of the environment at $t = 0$

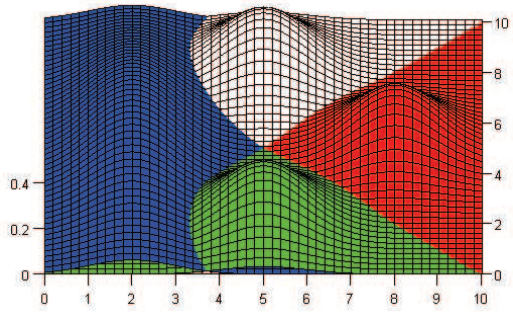

Fig. 2.b: Snapshot of the environment at $t = 1/3$

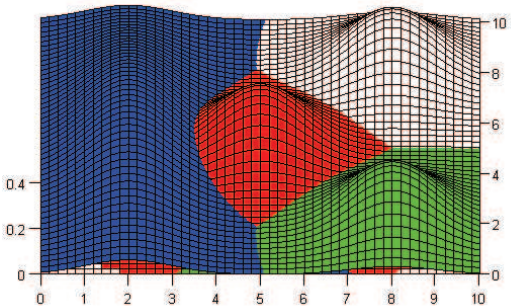

Fig. 2.c: Snapshot of the environment at $t = 2/3$

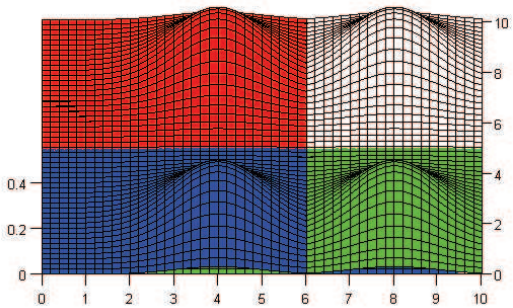

Fig. 2.d: Snapshot of the environment at $t = 1$

TABLE 1. PARAMETRIC EQUATIONS GOVERNING PATH OF DRIFT

	$t = 0$ to t = 1/3				$t = 1/3$ to t = 2/3				$t = 2/3$ to t = 1			
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y
C_1	2	5	1	1+6t	2	5	1	3	2+6(t-2/3)	5-9(t-2/3)	1	6(t-2/3)
C_2	8	5	1	1	8-9(t-1/3)	5	1	1	5-3(t-2/3)	5+9(t-2/3)	1	1
C_3	5	2	3-6t	1	5+9(t-1/3)	2	1	1	8	2	1	1
C_4	5	8	3-6t	1	5+9(t-1/3)	8	1	1	8	8	1	1

Figures 3~6 show the simulation results obtained on this environment. All classification performances were computed on the entire feature space, calculated with respect to individual instances' (likelihood) probability of occurrence (to prevent instances away from decision boundaries artificially increasing the performance, and to obtain a fare comparison to Bayes classifier). All results are average of 100 independent trials.

Figure 3 illustrates the classification performance of a single classifier (an MLP) for various values of rate-of-change (inversely proportional to parameter T) compared to that of the Bayes classifier. We include this rather crowded figure primarily to show that a single classifier performance – including the Bayes classifier – on the concept drift problem is independent of the rate of change, since there is only one classifier evaluated on the most current environment only. In subsequent figures, we plot the ensembles performances for various values T, however, we only plot one performance for single classifiers, to prevent figures from getting overcrowded.

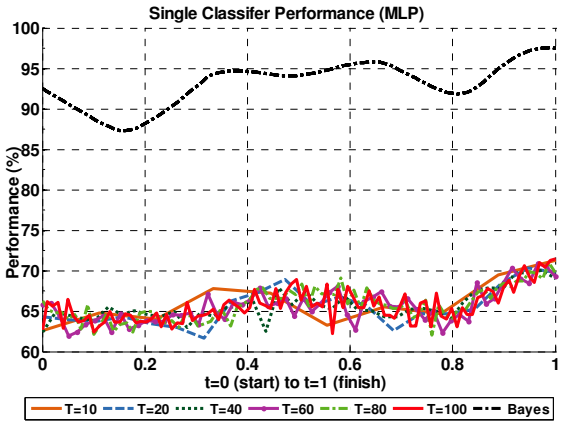

Fig.3 Single MLP performance for various values of T.

Figures 4~6 show Learn^{++}.NSE performances when trained with MLP, SVM and naïve Bayes classifiers, respectively. The top curve in each plot is the performance of the optimal Bayes classifier, statistically the best performance that can be achieved. The remaining curves are Learn^{++}.NSE performances for various T values, and that of the single classifier (usually the bottom-most curve). The same line styles are used in all figures.

Several observations can be made from these figures. First, the Learn^{++}.NSE ensemble follows the optimal Bayes classifier performance quite closely.

Second, the slower the rate of change, the higher the performance, and the closer is the ensemble performance to that of the Bayes classifier. In fact, there is a strong negative correlation between the rate of change and the performance. This makes intuitive sense, since a slower changing environment is easier to track. Furthermore, as T increases, the most recently generated previous classifiers in the ensemble become more relevant to the current environment, thus increasing the performance.

Third, in all cases, for all types of base models and rate-of-change values, the ensemble performs as good or better – and usually much better – than the single classifier. As mentioned earlier, this is not a trivial achievement, as the single classifier does not carry any information that is not relevant to the current environment,

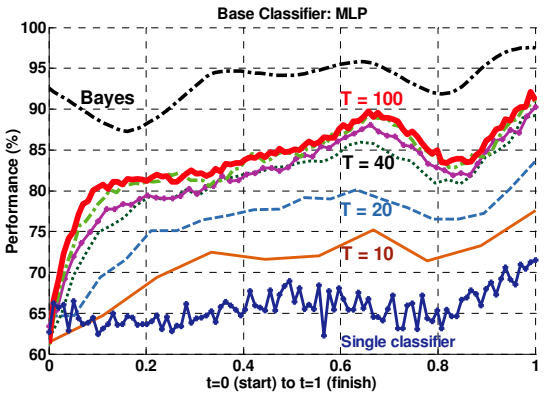

Fig. 4. Learn^{++}.NSE performance with MLP

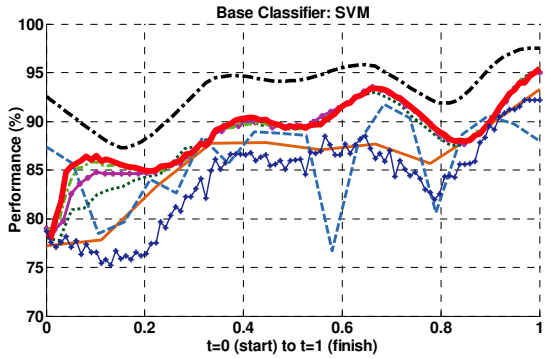

Fig. 5. Learn^{++}.NSE performance with SVM

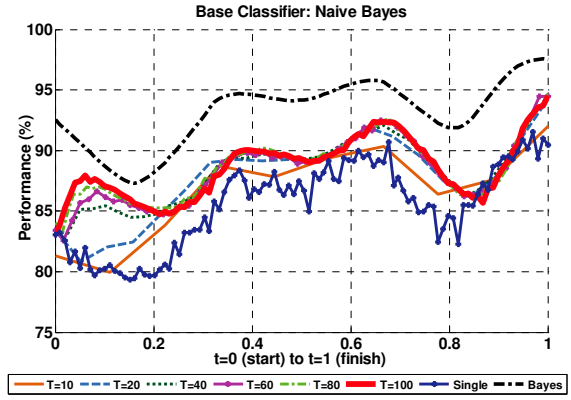

Fig. 6. Learn^{++}.NSE performance with naïve Bayes

whereas the older classifiers do. The ensemble performance was substantially better than that of the single classifier, particularly for MLP and SVM classifiers.

Fourth, the performance trends exhibited in all experiments are very similar, regardless of the base classifier used to generate ensemble members. In all cases, the ensemble performance matches or beats the single classifier performance – and usually by wide margins.

Note that the actual performance numbers – as promising as they may be – are not of primary importance in this experiment. For example, the apparent drops and increases in the performance throughout the experiment merely indicate that the underlying classification problem is getting increasingly easier or more difficult in time, as evidenced by the declining or improving optimal Bayes classifier performance. We are primarily con-

cerned with how well Learn^{++}.NSE tracks Bayes classifier, as we cannot expect any algorithm to do any better.

These observations indicate that the algorithm can indeed track the changing environment quite successfully, it is able to retain and extract information from previous classifiers, and able to do so using a variety of base classifiers, whether it is an online algorithm such as naïve Bayes, or an algorithm that can only be trained in batch mode, such as MLP or SVM.

C. *Experiment 2: Cyclic Environment*

As we discussed earlier, an important feature of Learn^{++}.NSE is its inclusion and exclusion of old classifiers through a strategic weighting mechanism. This mechanism allows older classifiers to be reused if the environment returns to an earlier state. To evaluate this aspect of the algorithm, we designed a second experiment where the data distribution eventually returns to its original state. Specifically, the experiment begins (at t = 0) with the mean of three different Gaussian distributions (of equal variance) located at their initial positions, as shown in Figure 7. The simulation involves each class traversing a triangular path in a clockwise direction. The final stage of the drift is reached at t = 1, when each class returns to its original starting position. Figure 8 shows the probability density distributions of all three classes at four different time instances. The drift path for this experiment is defined through the parametric equations shown in Table 2. As in the previous experiment, we have varied the number of time steps from start to finish using the array of T = {10, 20, 40, 60, 80, 100} values. All simulations were again repeated 100 times, and the performances presented below are averages of these 100 independent trials.

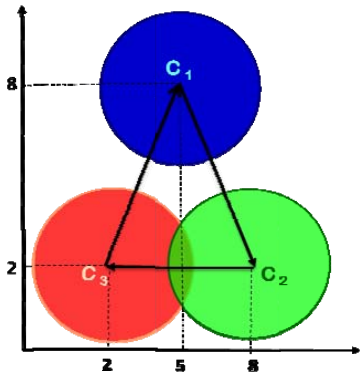

Fig. 7. Conceptual description of the drift path for the cyclic environment

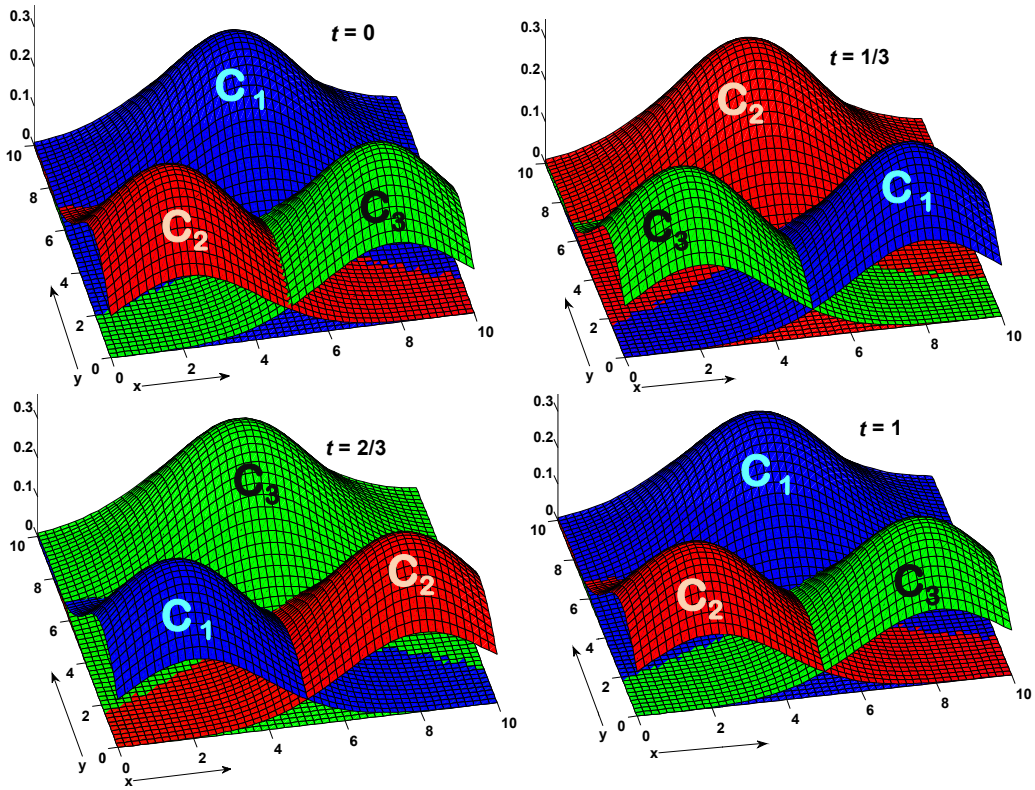

Fig. 8. Snapshots of the environment showing current distributions at time t = 0, t = 1/3, t = 2/3 and t = 1.

TABLE 2. PARAMETRIC EQUATIONS GOVERNING THE PATH OF DRIFT FOR THE CYCLIC ENVIRONMENT DRIFT

	t = 0 to t = 1/3				t = 1/3 to t = 2/3				t = 2/3 to t = 1			
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y
C_1	5+9t	8-18t	2	2	8-18(t-1/3)	2	1	3	2+9(t-2/3)	2+18(t-2/3)	2	2
C_2	2+9t	2+18t	2	2	5+9(t-1/3)	8-18(t-1/3)	1	1	8-18(t-2/3)	2	2	2
C_3	8-18t	2	2	2	2+9(t-1/3)	2+18(t-1/3)	1	1	5+9(t-2/3)	8-18(t-2/3)	2	2

The simulation results comparing the Learn^{++}.NSE ensemble performance to that of Bayes classifier and the single classifier are shown in Figures 9, 10, and 11 for MLP, SVM and naïve Bayes base classifiers, respectively. The same formatting and line styles are used as before.

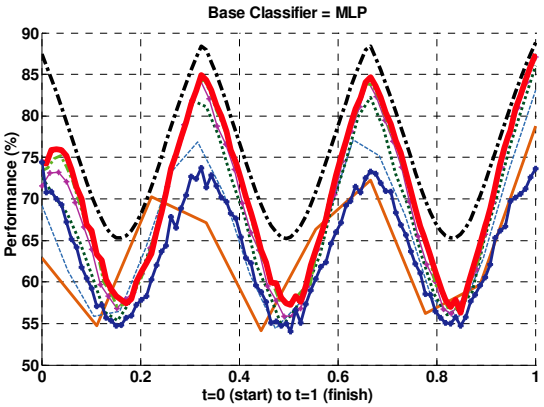

Fig. 9. Learn^{++}.NSE performance with MLP

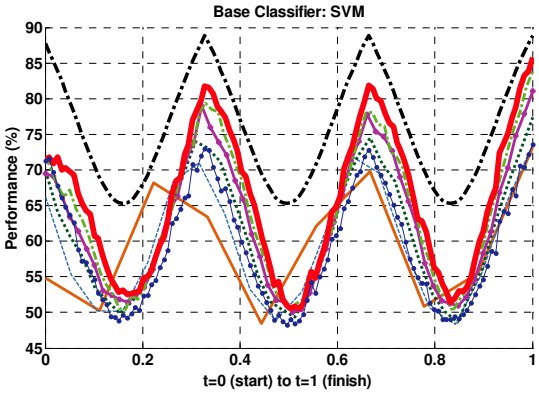

Fig. 10. Learn^{++}.NSE performance with SVM

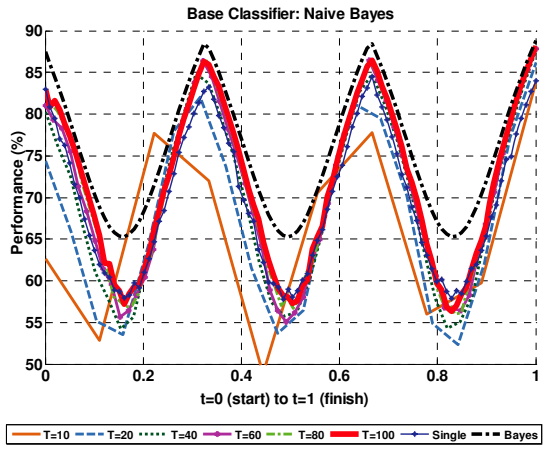

Fig. 11. Learn^{++}.NSE performance with naïve Bayes

In general, the observations made in Experiment 1 can also be seen in this experiment: the proposed algorithm very closely follows the Bayes performance, with increasing performance for increasing T values. The algorithm also matches or exceeds – and typically with wide margins – the performance of the single classifier of the same type used in generating the ensemble, regardless of the base classifier.

However, the cyclical nature of the environment reveals additional observations. Note that while the environment returns to its original state only at t = 1, the symmetric nature of the problem renders a sinusoidal performance characteristic. This can be easily seen from Figure 8, where the class conditional probabilities receive identical and rotating values at t = 0, t = 1/3, t = 2/3 and t = 1. For example, if class conditional probabilities are $P(\boldsymbol{x}, y|C_1) = a, P(\boldsymbol{x}, y|C_2) = b,$ and $P(\boldsymbol{x}, y|C_3) = c$ at t = 0, then these probabilities will be $P(\boldsymbol{x}, y|C_1) = c,$ $P(\boldsymbol{x}, y|C_2) = a,$ and $P(\boldsymbol{x}, y|C_3) = b$ at t = 1/3, and $P(\boldsymbol{x}, y|C_1) = b,$ $P(\boldsymbol{x}, y|C_2) = c,$ and $P(\boldsymbol{x}, y|C_3) = a$ at t = 2/3. Therefore, the Bayes error at these time instances will all be identical, explaining the sinusoidal nature of performance plots. Another interesting observation can be made at t = 1, where the class distributions return to their starting values of t = 0. As mentioned above, the Bayes error at t = 1 is identical to those at t = 0, t = 1/3 and t = 2/3. The Learn^{++}.NSE performances, while equal at t = 1/3 and t = 2/3, are not identical at t = 0 and t = 1. Specifically, the algorithm's performance is relatively low at the first few time steps after t = 0; this is because there are very few classifiers in the ensemble. However, the algorithm quickly catches up and starts closely following the performance of Bayes classifier. On the other hand, at t = 1, Learn^{++}.NSE exceeds its t = 1/3 and t = 2/3 performances and virtually reaches the Bayes performance. This is because, the algorithm can now take advantage of its earliest classifiers, which were trained on data drawn from a very similar distribution to the current one. The weighting mechanism realizes that those classifiers carry relevant information and uses them in the ensemble by assigning them higher weights.

IV. CONCLUSIONS

We have outlined an ensemble of classifiers based algorithm that can learn in non-stationary environments. When presented with data from then current snapshot of a drifting environment, Learn^{++}.NSE creates a new classifier that is added to the ensemble. The classifiers are then combined using dynamically weighted majority voting, where each classifier's weight is determined by its error, age, and performance on current and all previous environments. The algorithm learns in an incremental fashion, that is, it does not use or store any of the previous datasets, but rather just the classifiers' performances on earlier environments. Unlike many other

concept-drift algorithms, Learn++.NSE does not discard any of the classifiers, but rather temporarily lowers their voting weights or even suspends them in proportion of their performance on the current environment. This weighting mechanism provides a delicate stability- plasticity balance.

On two experiments, we showed that the algorithm can track the changing environments regardless of the rate of change, though – as expected – the performance markedly increases for slower rates of change. The experiments used synthetically generated data drawn from Gaussian distributions so that the performance of the algorithm could be compared to that of the optimal Bayes classifier. The simulation results indicate that the algorithm is able to track the changing environment and follow the Bayes classifier performance. The ensemble always meets or exceeds the single classifier trained on the most recent data. The algorithm can also work with any supervised base classifier, whether that classifier can learn in an online fashion or not.

This algorithm is not intended for all concept drift problems. Specifically, we expect Learn^{++}.NSE to perform well when the data available from each snapshot cannot adequately describe the environment, and when there is overlap in data distributions over consecutive time instances. Under these relatively mild conditions, a portion of the earlier classifiers will always carry relevant information and help improve the performance of the ensemble on the current environment.

Finally, we note that retaining all classifiers come with its own price tag: the memory requirement necessary to store the parameters of all classifiers, some of which may be dormant at any time. Given the current availability of low cost memory, and that we only save the classifier parameters and not the entire data, we believe this is a very reasonable price to pay.

This algorithm is currently in its early stages of its development; however, initial results are promising and warrant further analysis of this approach. Our current and future work includes evaluating the algorithm on higher dimensional datasets, including those obtained from real world problems.

REFERENCES

[1] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural Networks*, vol. 1, no. 1, pp. 17-61, 1988.

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

[3] A. Kuh, T. Petsche, and R. L. Rivest, "Learning time-varying concepts," *Advances in Neural Information Processing*, San Mateo, CA: Morgan Kaufmann, 1991, pp. 183-189.

[4] J. C. Schlimmer and R. H. Granger, "Incremental Learning from Noisy Data," *Machine Learning*, vol. 1, no. 3, pp. 317-354, Sept.1986.

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

[6] R. Klinkenberg and T. Joachims, "Detecting concept drift with support vector machines," *17th International Conference on Machine Learning*, 2000, pp. 487-494.

[7] R. Klinkenberg, "Learning Drifting Concepts: Example Selection vs. Example Weighting," *Intelligent Data Analysis, Incremental Learning Systems Capable of Dealing with Concept Drift*, vol. 8, no. 3, pp. 281-300, 2004.

[8] M. A. Maloof and R. S. Michalski, "Incremental learning with partial instance memory," *Artificial Intelligence*, vol. 154, no. 1-2, pp. 95-126, Apr.2004.

[9] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with Drift Detection," *Advances in Artificial Intelligence - SBIA 2004*, Lecture Notes in Computer Science, vol. 3171, 2004, pp. 286-295.

[10] P. Vorburger and A. Bernstein, "Entropy-based Concept Shift Detection," *6th Int. Conference on Data Mining (ICDM '06.)*, 2006, pp. 1113-1118.

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

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

[13] A. Tsymbal, "Technical Report: The problem of concept drift: definitions and related work," Trinity College, Dublin, Ireland,TCD-CS-2004-15, 2004.

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

[15] A. Blum, "Empirical Support for Winnow and Weighted-Majority Algorithms: Results on a Calendar Scheduling Domain," *Machine Learning*, vol. 26, no. 1, pp. 5-23, Jan.1997.

[16] N. Oza, "Online Ensemble Learning." Ph.D. Dissertation, University of California, Berkeley, 2001.

[17] K. Nishida, K. Yamauchi, and O. Takashi, "ACE: Adaptive Classifiers-Ensemble System for Concept-Drifting Environments," *Multiple Classifier Systems*, Lecture Notes in Computer Science ,N. Oza, R. Polikar, J. Kittler, and F. Roli, Eds. vol. 3541, 2005, pp. 176-185.

[18] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: a new ensemble method for tracking concept drift," *3rd IEEE Int. Conf. on Data Mining (ICDM 2003)*, 2003, pp. 123-130.

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

[20] D. Muhlbaier and R. Polikar, "An Ensemble Approach for Incremental Learning in Nonstationary Environments," *7th. Int. Workshop on Multiple Classifier Systems*, Lecture Notes in Computer Science, vol. 4472, Berlin: Springer, 2007, pp. 490-500.

[21] M. D. Muhlbaier and R. Polikar, "Multiple Classifiers Based Incremental Learning Algorithm for Learning in Nonstationary Environments," Int. Conf. on *Machine Learning and Cybernetics,* vol. 6, 2007, pp. 3618-3623.

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

[23] R. Polikar, L. Udpa, S. S. Udpa, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 31, no. 4, pp. 497-508, 2001.