Incremental Learning of Concept Drift in Nonstationary Environments

Ryan Elwell, *Member, IEEE*, and Robi Polikar, *Senior Member, IEEE*

***Abstract*— We introduce an ensemble of classifiers-based approach for incremental learning of concept drift, characterized by nonstationary environments (NSEs), where the underlying data distributions change over time. The proposed algorithm, named Learn^{++}.NSE, learns from consecutive batches of data without making any assumptions on the nature or rate of drift; it can learn from such environments that experience constant or variable rate of drift, addition or deletion of concept classes, as well as cyclical drift. The algorithm learns incrementally, as other members of the Learn^{++} family of algorithms, that is, without requiring access to previously seen data. Learn^{++}.NSE trains one new classifier for each batch of data it receives, and combines these classifiers using a dynamically weighted majority voting. The novelty of the approach is in determining the voting weights, based on each classifier's time-adjusted accuracy on current and past environments. This approach allows the algorithm to recognize, and act accordingly, to the changes in underlying data distributions, as well as to a possible reoccurrence of an earlier distribution. We evaluate the algorithm on several synthetic datasets designed to simulate a variety of nonstationary environments, as well as a real-world weather prediction dataset. Comparisons with several other approaches are also included. Results indicate that Learn^{++}.NSE can track the changing environments very closely, regardless of the type of concept drift. To allow future use, comparison and benchmarking by interested researchers, we also release our data used in this paper.**

***Index Terms*— Concept drift, incremental learning, learning in nonstationary environments, multiple classifier systems.**

I. Introduction

MUCH of the recent history of machine learning research has focused on learning from data assumed to be drawn from a fixed yet unknown distribution. Learning in a nonstationary environment (or learning *concept drift*), where the underlying data distribution changes over time; however, has received much less attention despite the abundance of applications that generate inherently nonstationary data. While algorithms for learning in such environments have recently started to appear in the literature, many make restrictive assumptions such as assuming slow or gradual drift, noncyclical environments, no new classes, partial availability of old data, or have not been tested on meaningful and truly nonstationary real world test beds. Yet, if the ultimate goal of computational intelligence is to learn from large volumes of data that come from real applications, then the need for a general framework for learning from—and adapting to—a nonstationary environment can be hardly overstated. Given new data, such a framework would allow us to learn any novel content, reinforce existing knowledge that is still relevant, and forget what may no longer be relevant, only to be able to recall, if and when such information becomes relevant again in the future.

This paper describes such a framework and proposes an incremental learning algorithm, Learn^{++}.NSE, that does not put restrictions on how slow, rapid, abrupt, gradual, local, global, cyclical or otherwise changes in distributions may be, whether new data introduce new concept classes or remove old ones, or whether old data are still relevant or even available. Learning new data in the absence of old data requires incremental learning, which raises the so-called stability–plasticity dilemma, where "stability" describes retaining existing (and still relevant or recurring) knowledge and "plasticity" refers to learning new knowledge [1]. We show that learning in such an environment and obtaining a meaningful stability–plasticity balance can be achieved by a strategic combination of an ensemble of classifiers that use dynamically assigned weights.

Interestingly, we also show that the proposed framework is consistent with the existing models of human learning, such as the Schema [2], [3], and Scaffolding Theory [4]. Schema describes a body of knowledge that is continually updated and modified as information is acquired through new experiences, given that current and prior knowledge may conflict. Scaffolding describes the role of a supervisor in monitoring incoming data and learner's performance to improve the learning process.

We organize our discussion as follows. We introduce the human learning theory in Section II, followed by an overview of learning concept drift in Section III. The Learn^{++}.NSE algorithm is introduced in Section IV, followed by a description of real and synthetic datasets used to evaluate the algorithm on various drift scenarios, as well as the comparative results of Learn^{++}.NSE and other approaches on these scenarios in Section V. Discussions and concluding remarks are provided in Section VI.

II. Human Learning

Knowledge acquisition is fundamental to both human and machine learning, and since the brain is often confronted with new environments containing information that may conflict with its prior knowledge or experience, connections between

Manuscript received July 31, 2010; revised January 7, 2011, and April 6, 2011; accepted June 4, 2011. Date of publication August 4, 2011; date of current version October 5, 2011. This work was supported by the National Science Foundation under Grant ECCS 0926159.

The authors are with the Signal Processing & Pattern Recognition Laboratory, Electrical & Computer Engineering Department, Rowan University, Glassboro, NJ 08028 USA [e-mail: ryan.elwell@gmail.com; (Corresponding author: polikar@rowan.edu)].

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2011.2160459

machine and human learning can provide guidelines for developing concept drift algorithms. In this section, we briefly review the two main theories of human leaning, and point out the similarities and connections to machine learning.

A. Schema Theory

Piaget asserts that an equilibrium, similar to that seen in physical processes, also applies to human cognition, describing the learning process as a constant effort to maintain or achieve balance between prior and new knowledge [5], [6]. The machine learning counterpart (henceforth indicated by $\leftrightarrow$ notation) is of course the [$\leftrightarrow$ stability–plasticity dilemma] [1]. Piaget's model forms the basis of a foundational theory for human learning, and has been extensively researched, specifically with regard to the *schema* theory [2], [3].

Schema theory is a psychological model that describes the process of human knowledge acquisition and memory organization for future decision-making. Two properties of this process are *schemata construction* [$\leftrightarrow$ incremental learning], which is building and categorization of the knowledge base as new information become available, and *schemata activation*, [$\leftrightarrow$ evaluation/generalization], the utilization of schema to interpret unknown or novel information.

Schemata construction is the process of building a knowledge base that can adapt to new information, which may or may not be consistent with the prior knowledge. The conflicts between the two are the building blocks of human (and computer) learning. Three terms are used to describe how such conflicts are handled. *Accretion* occurs when information is remembered or interpreted in the context of existing schema, i.e., when new information is agreeable with the current body of knowledge. Minor differences between incoming information and prior knowledge often necessitate *tuning* of (or *assimilation* into) the schema. When new knowledge cannot be accommodated under existing schema because of severe conflict, the result is *restructuring* (or *accommodation*) to create new schemata that supplements or replaces the prior knowledge base. The machine learning counterpart of schemata construction is [$\leftrightarrow$ incremental learning], which also needs to address the same accretion, tuning and restructuring issues.

Schemata activation occurs for two reasons. First, schemata are activated during the data acquisition process in order to determine which type of schemata construction (tuning, accretion, restructuring) should take place. Here, the current knowledge base must be evaluated and compared to new knowledge in order to make connections and determine its adequacy to handle or understand the new information.

Second, schemata are activated for prediction and extrapolation, enabling the brain to interpret novel data and make predictions; naturally, such predictions are based on current schemas. Not only can the brain interpret novel information, but it can also hypothesize about missing material within its own knowledge base, hence the brain is quite robust in the presence of structural damage that leads to memory loss.

The counterpart of activation in machine learning is evaluation (validation) of the model performance to iteratively fine tune the next step of learning as new data become available.

B. Scaffolding Theory

Scaffolding is a tutoring theory developed to enhance human learning of complex data [4]. Scaffolding is a supervised learning approach to build schemata by breaking up complex information such that it is learned in chunks [$\leftrightarrow$ batch learning], and by periodically intervening to evaluate performance within the scope of the most recent information. As we describe in Section IV, this is precisely what Learn^{++}.NSE does as it updates the ensemble, one classifier at a time, based on the current and previous errors on the current batch of data.

The goal of scaffolding is to provide a learner with both *feedback* and *guidance*. Passive supervision provides the learner with experience–consequence combinations [$\leftrightarrow$ features–correct labels] as knowledge [$\leftrightarrow$ data] become available; whereas active scaffolding enhances learning by *complexity reduction* [$\leftrightarrow$ preprocessing, feature selection], *problematizing* [7], i.e., finding conflicts between current and prior knowledge [$\leftrightarrow$ drift detecting], and *fading* [7], [8], preventing redundancy when an environment has been learned [$\leftrightarrow$ pruning].

III. Concept Drift/Non-Stationary Learning

A. Definitions

Informally, *concept drift* refers to a change in the class (concept) definitions over time, and therefore a change in the distributions from which the data for these concepts are drawn. An environment from which such data is obtained is a *non-stationary* environment. Starting with the Bayes posterior probability of a class that a given instance belongs, $P(\omega|x) = P(x|\omega)P(\omega)/P(x)$, concept drift can be formally defined as any scenario where the posterior probability changes over time, i.e., $P_{t+1}(\omega|x) \neq P_t(\omega|x)$. An in-depth look at this fundamental definition is important to understand the different aspects of concept drift.

$P(x)$ describes the feature-based probabilities (evidence) of the data. Observing $P(x)$ over time allows us to see general changes in the environment that generates this data. Although a change in overall distribution of the features often means that the true decision boundaries are shifting as well, an *observation of change* in $P(x)$ is insufficient to definitively indicate concept drift because of its independence of the class labels.

$P(x|\omega)$ describes the likelihood of observing data point x within a particular class. This likelihood measurement is a class-dependent probability and is governed by the previously seen data instances. A shift in likelihood would seem to indicate that the class labels may also be changing. However, we assert that it is not until the distribution of one class shifts such that the true class boundaries are altered that we can call this change a real concept drift. Class drift without overlapping of true class boundaries is known as *virtual* concept drift [9], and merely shows that the learner is being provided with additional data from the *same* environment. Virtual drift is the result of an incomplete representation of the true distribution in the current data. The key difference is that real drift requires replacement learning (where old knowledge becomes irrelevant [restructuring]), whereas virtual drift requires supplemental learning (adding to the current knowledge [$\leftrightarrow$ tuning]).

Finally, $P(\omega)$, defines class prior probabilities, and relates class balance to the overall distribution. Since there is no relation to the features, observing $P(\omega)$ does not reveal information about the decision boundaries between classes. Yet it does reveal another fundamental aspect of non-stationary environments dealing with class imbalance, as class imbalance is known to negatively impact classification performance [10], [11].

Concept drift can also be viewed in a more abstract sense as an obstacle caused by insufficient, unknown or unobservable features in a dataset, a phenomenon known as *hidden context* [12]. In such a case, there is an underlying phenomenon that provides a true and static description over time for each class, which, unfortunately, is hidden from the learner's view. Viewing the problem with the benefit of this (hidden) context would remove the non-stationarity. Yet, the learner must cope with the available information. Since we can never know the hidden context, we use the aforementioned probabilistic definition of concept drift to describe nonstationary environments.

Quinonero-Candela [11], Minku [13], and Kuncheva [14], [15] also provided comprehensive summaries for characterizing different types of concept drift with respect to its *speed*, *randomness*, and *cyclical* nature. Drift speed describes the displacement rate in $P_t(\omega|x)$ from one time step to the next, $P_{t+1}(\omega|x)$. Larger displacement within a step denotes fast drift and usually results in high classifier error. Gradual drift, however, appears in smaller displacements, results in lower classification error, and as a result, is more difficult to detect.

Drift randomness is an important descriptor in discerning between non-stationary and noisy data, and can be described as the variance of a distribution over a short period time. Randomness can be viewed in terms of its frequency and magnitude: high variance between two periods of time indicates a highly unstable environment which, as this level increases, approaches a state where the environment cannot be learned.

The cyclical nature of drift is a phenomenon that can be observed in many real-world applications such as climate or electricity demand modeling. In such cases, class definitions change in such a way that a previous environment may recur after some period of time. This recurrence can be periodic or random.

Finally, addition or deletion of new concepts is typically not addressed by concept drift algorithms; as such events are characterized more by *concept change* rather than concept drift. Hence, we use the more general terminology of *learning in nonstationary environments* or *nonstationary learning* (NSL) to refer to any drift or change regardless of its nature. The framework proposed in this paper addresses all issues of NSL, not just those associated with concept drift.

B. Desired Properties of Concept Drift/NSL Algorithms

Combining schema and scaffolding theories, Kuncheva's suggested desiderata for learning concept drift [15], and the generally accepted definitions of incremental learning [16], [17], we use the following guidelines to develop a framework for learning in non-stationary environments. 1) Any given instance of data can only be seen once for the purpose of training; therefore knowledge from each instance must be generalized, summarized or stored in some way in the model parameters for future use. This requires a truly incremental (or one-pass) learning, where previous data may not be used for future training. 2) Since the most recent dataset is a representation of the current environment, knowledge should be categorized based on its relevance to the current environment, and be dynamically updated as new data become available. 3) The learner should have a mechanism to reconcile when existing and newly learned knowledge conflict with each other. More specifically, there should be a mechanism for monitoring both the incoming data and the learner's performance on new and old data for the purpose of *complexity reduction*, *problematizing*, and *fading*. 4) The learner should have a mechanism to forget or discard information that is no longer relevant, but preferably with the added ability to recall such information if the drift or change follow a cyclical nature. 5) Knowledge should be incrementally and periodically stored so that it can be activated to produce the best hypothesis for an unknown (unlabelled) data instance at any time during the learning process.

C. Review of Existing Approaches

NSL has recently been receiving increasing attention, in part due to many practical applications, such as spam, fraud or climate change detection, where data distributions inherently change over time. Algorithms designed for concept drift can be characterized in several ways, such as online versus batch algorithms; single classifier versus ensemble-based approaches; or active versus passive approaches, with active approaches featuring a drift detection mechanism, learning only when drift is detected. Passive approaches, on the other hand, assume possibly ongoing drift and continuously update the model with each new data(set).

Online algorithms learn one instance at a time, whereas batch learning requires blocks of instances. Online learners have better plasticity but poorer stability properties. They also tend to be more sensitive to noise as well as to the order in which the data are presented. Batch learners benefit from the availability of larger amounts of data, have better stability properties, but can be ineffective if the batch size is too small, or if data from multiple environments are present in the same batch. Most batch learners of concept drift typically use some form of windowing to control the batch size. Earliest examples of this—also called *instance selection*—approach include single classifier, passive batch algorithms STAGGER [18] and FLORA [12], which use a sliding window to choose a block of (new) instances to train a new classifier. The window size is modified via "window adjustment heuristic," based on how fast the environment is changing. FLORA has a built-in forgetting mechanism with the implicit assumption that those instances that fall outside the window are no longer relevant, and the information carried by them can be forgotten. More recently, there have been several additions to this window-based approach, each introducing its own heuristics on drift detection [19], choice of classifier (such as decision trees, fuzzy rules, kNN, etc.) [20], [21], or establishing error thresholds [22]. The primary shortcoming of these approaches is that they are often not incremental (they need access to old data), or cannot handle cyclic environments. Other single classifier active approaches typically include novelty (anomaly) detection to determine when changes occur, e.g., by using control charts-based CUSUM [23], [24], confidence interval on error [22], [25], other statistical approaches [26], treating NSL as a prediction problem [27], or deriving online update rules based on minimization of a penalty function for the perceptron [28], [29]. Another group of approaches use information theoretic measures, e.g., entropy, mutual information or Hoeffding bounds of individual features for detecting drift and updating a decision tree [30]–[32]. Many of these approaches also include a FLORA-like windowing mechanism, so do Hulten *et al.*'s concept adapting very fast decision tree [33] or Cohen *et al.*'s incremental online-information network [25], [34] algorithms. Bayesian or Kalman filter-based approaches for online update of model parameters have also been proposed, e.g., linearly separable rules [28], for regression [35], or semi-supervised learning problems [36].

The ensemble-based approaches that combine multiple classifiers constitute a new breed of NSL algorithms. Kuncheva puts ensemble-based approaches into one of the three general categories [15]: given new data, those that: 1) update the combination rules or voting weights of a fixed ensemble, such as [37], [38]; the origins of which can be traced to Littlestone's Winnow [39] and Freund and Schapire's Hedge (a precursor of AdaBoost) [40]; 2) update the parameters of existing ensemble members using an online learner [22], [41], and/or 3) add new members to grow an ensemble with each incoming dataset. The latter category approaches typically use a passive drift detection along with fixed ensemble size, where the oldest (as in Street's Streaming Ensemble Algorithm (SEA) [42], and Chen and He's Recursive Ensemble Approach (REA) [43]) or the least contributing ensemble members are replaced with a new one (as in Tsymbal's Dynamic Integration [44], Kolter and Maloof's online algorithm, Dynamic Weighted Majority (DWM) [45]). While most ensemble approaches use some form of voting, there is disagreement on the type of voting to be used. For example, Tsymbal relates classifier weight to performance as well as a proximity factor, giving higher weight to a classifier if its training data were in the same region as the testing example [44], whereas Gao—indicating that weights based on classifier error on data whose distribution changes is uninformative for future datasets—prefers a simple (unweighted) majority vote [46].

Other efforts that follow similar ensemble approaches include [47]–[50], as well as hybrid approaches such as random forests with entropy [51], and Bifet's Hoeffding tree with Kalman filter-based active change detection using adaptive sliding window (ADWIN) [52], [53]. ADWIN is also available within the WEKA-like software suite, massive online analysis at [54]. Alternatively, Scholz and Klinkenberg's approach maintains two ensembles—one trained on the current data, and one trained on a cache of previous data (hence nonincremental), and chooses the better of the two ensembles in each time step. Classifier weights are based on the "LIFT" of each classifier, measuring the correlation between the classifier's decision and the true class based on conditional probabilities. Varying LIFT values for a classifier across time indicate the existence of drift [55]. The way in which the LIFT values are computed, however, restricts the algorithm to binary classification problems only.

D. Drift Detection

We conclude this section with a short discussion on drift detection. Recall that the main goal of the supervisor in scaffolding theory of human learning is to provide guidance and feedback to the learner by: 1) problematizing data (determining and/or removing conflicts between incoming data and the current knowledge base); 2) simplifying complex data; and 3) fading, i.e., ceasing the learning process when an environment has been learned. Each of these tasks requires some level of feedback about the incoming data or the learner's performance at any given time. This feedback is used to discern how the new information differs from previous data, and determine the learner's capability to grasp current concepts. In computer learning of nonstationary environments, the corresponding problem is to determine when the environment has sufficiently changed such that the existing models can no longer explain the current data. Determining when such a change, i.e., whether concept drift has occurred, is known as *drift detection*.

As mentioned above, concept drift algorithms can be *active* or *passive* with respect to the drift detection mechanism. An active drift detection method seeks to pinpoint the time and severity of the drift, and allow the classifier to modify or continue learning accordingly. Hence, active learning integrates all scaffolding techniques to fine-tune the learner's plasticity. A significant downside of active learning, however, is the risk of having an imperfect detection mechanism which may—and often does—yield false reports, an all too common occurrence particularly for noisy datasets. In *passive* drift detection, however, the learner acknowledges that the environment may change at any time or may be continuously changing. The algorithm then continually learns from the environment by constructing and organizing the knowledge base. If change has occurred, this change is learned. If change has not occurred, existing knowledge is reinforced.

IV. Learn^{++}.NSE

A. Background: The Learn^{++} Family of Algorithms

The proposed algorithm, Learn^{++}.NSE, is a member of the Learn^{++} family of algorithms. The common denominator in all Learn^{++} algorithms is an ensemble of classifiers that are incrementally trained (with no access to previous data) on incoming batches of data, and combined with some form of weighted majority voting. The distribution update rule for choosing data for training subsequent ensemble members, and the mechanism for determining the voting weights are the distinguishing characteristics of different Learn^{++} algorithms. The original Learn^{++} [16] is an AdaBoost-like algorithm for learning from a stationary distribution from which data are incrementally acquired in batches. Learn^{++}.NC [17] was later developed for learning New Classes (NC), with new data from existing classes assumed to remain stationary. Learn^{++}.NC employs a dynamically weighted consult-and-vote mechanism to determine which classifiers should or should not vote for any given instance based on the (dis)agreement among classifiers trained on different classes. In Learn^{++}.MF, ensemble members are trained on different subsets of the features, so that Missing Features (MF) can be accommodated by combining ensemble members trained on the currently available features [56]. While all former Learn^{++} algorithms do some form of incremental learning, none of them is capable of learning from a nonstationary environment, and Learn^{++}.NSE is developed specifically to fill this gap.

Preliminary versions of Learn^{++}.NSE and its early results have appeared in conference proceedings, such as [57]–[60]. These efforts primarily investigated the impact of: 1) the type of base classifier [MLP versus SVM. versus Naïve Bayes(NB)]; 2) the rate of drift (e.g., slow versus rapid); and 3) pruning (whether age or error-based pruning improve performance). Based entirely on synthetic data experiments, we found that the Learn^{++}.NSE is generally independent of the base classifier, works best if no pruning is used (particularly for recurrent environments), and as expected, the slower the drift, the better the environment can be tracked. In this paper, we formally introduce the algorithm in detail, show that it works on a variety of concept drift scenarios, including variable drift and class addition/removal, provide a very informative analysis of voting weights, and compare Learn^{++}.NSE to several existing popular approaches on carefully designed synthetic as well as real world data.

```
**Input:** For each dataset $\mathcal{D}^t$   $t = 1,2,\ldots$
Training data $\{x^t(i) \in X;\ y^t(i) \in Y = \{1,\ldots,c\}\},\ i = 1,\ldots,m^t$
Supervised learning algorithm **BaseClassifier**
Sigmoid parameters $a$ (slope) and $b$ (infliction point)
**Do for** $t = 1,2,\ldots$
    **If** $t = 1$, **Initialize** $D^1(i) = w^t(i) = 1/m^1, \forall i,$                (1)
    Go to step 3. **Endif**
    1. Compute error of the existing ensemble on new data
        $E^t = \Sigma_{i=1}^{mt} 1/m^t \cdot [\![H^{t-1}(x^t(i)) \neq y^t(i)]\!]$          (2)
    2. Update and normalize instance weights
        $w_i^t = \frac{1}{m^t} \cdot \begin{cases} E^t, & H^{t-1}(x^t(i)) = y^t(i) \\ 1, & otherwise \end{cases}$   (3)
        Set $\boldsymbol{D}^t = \boldsymbol{w}^t / \Sigma_{i=1}^{mt} w^t(i) \Rightarrow D^t$ is a distribution   (4)
    3. Call **BaseClassifier** with $\mathcal{D}^t$, obtain $h^t: X \rightarrow Y$
    4. Evaluate all existing classifiers on new data $\mathcal{D}^t$
        $\varepsilon_k^t = \Sigma_{i=1}^{m^t} D^t(i) [\![h_k(x^t(i)) \neq y^t(i)]\!]$ 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 $k = 1,\ldots,t \rightarrow 0 \leq \beta_k^t \leq 1$   (6)
    5. Compute the weighted average of all normalized
        errors for $k^{th}$ classifier $h_k$: For $a,b \in \mathbb{R}$
        $\omega_k^t = 1/(1 + e^{-a(t-k-b)})$, $\omega_k^t = \omega_k^t / \Sigma_{j=0}^{t-k} \omega_k^{t-j}$   (7)
        $\overline{\beta_k^t} = \Sigma_{j=0}^{t-k} \omega_k^{t-j} \beta_k^{t-j}$, for $k = 1,\ldots,t$   (8)
    6. Calculate classifier voting weights
        $W_k^t = \log(1/\overline{\beta_k^t})$,   $for\ k = 1,\ldots,t$   (9)
    7. Obtain the final hypothesis
    → $H^t(x^t(i)) = \arg\max_c \Sigma_k W_k^t \cdot [\![h_k(x^t(i)) = c]\!]$   (10)
```

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

B. Algorithm Overview

Learn^{++}.NSE is an ensemble-based batch learning algorithm that uses weighted majority voting, where the weights are dynamically updated with respect to the classifiers' time-adjusted errors on current and past environments. It employs a passive drift detection mechanism, and uses only current data for training. It can handle a variety of nonstationary environments, including sudden concept change, or drift that is slow or fast, gradual or abrupt, cyclical, or even variable rate drift. It is also one of the few algorithms that can handle concept addition (new class) or deletion of an existing class.

The algorithm is provided with a series of training datasets $\mathcal{D}^t = x^t(i)\epsilon X; y^t(i)\epsilon Y,\ i = 1,\ldots,m^t$, where t is a time index. Hence, $x^t(i)$ is the i^{th} instance of the dataset (environment), drawn from an unknown distribution $P^t(x,y)$, which is the current snapshot of a possibly drifting distribution at time t. At time $t+1$, we obtain a new batch of data drawn from $P^{t+1}(x,y)$. At each time step there may or may not have been a change in the environment, and if there was, the rate of this change is also not known, nor assumed to be constant. Furthermore, we presume all 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. Hence, we ask the algorithm to work in a truly incremental fashion. Any information previously provided by earlier data must necessarily be stored in the parameters of the previously generated classifiers.

Depending on the nature of drift/change, Learn^{++}.NSE retains [$\leftrightarrow$ accretion], constructs or (temporarily) discards knowledge [$\leftrightarrow$ tuning, restructuring], so that it can be properly categorized [$\leftrightarrow$ activation] when asked to identify new data.

The knowledge base is initialized by creating a single classifier on the first available batch of data. Once prior knowledge is available, the current ensemble [$\leftrightarrow$ the knowledge base] is evaluated on the new data (Step 1 in Fig. 1). In Step 2, the algorithm identifies which examples of the new environment are not recognized by the existing knowledge base [$\leftrightarrow$ problematizing]. The knowledge base is updated [$\leftrightarrow$ restructuring] in Step 3, by adding a new classifier trained on the current training data. In Step 4, each classifier (including the one that has just been created) is evaluated on the training data. As previously unknown data have been identified in Step 2, the penalty for misclassifying such data is reduced in the error calculation. In other words, more credit is given to classifiers capable of identifying previously unknown instances, while classifiers that misclassify previously *known* data are penalized. In Step 5, classifier error is weighted with respect to time so that recent competence (error rate) is considered more heavily for categorizing knowledge. Voting weights are determined in Step 6 as log-normalized reciprocals of the weighted errors: if a particular classifier's knowledge does not match the current environment, that classifier receives little or no weight, and is effectively—but only temporarily—removed from the knowledge base. The classifier is not discarded: if its knowledge becomes relevant again, it is recalled through higher voting weights it receives on the then current environment. Learn^{++}.NSE will only forget temporarily, which is particularly useful in cyclical environments. The final decision is obtained in Step 7 as the weighted majority voting of the current ensemble members.

C. Algorithm Description

As $\mathcal{D}^t$ becomes available at time t, Learn^{++}.NSE is presented with $x^t(i), i = 1, \ldots, m^t$ instances and corresponding class labels $y^t(i)$ At $t = 1$, *instance specific error weights* of the first batch of data, $w^t(i)$, and a *penalty distribution* $D^1(i)$ are initialized to be uniform [as in (1)]. For all subsequent time steps, these quantities are initialized based on the error of the existing ensemble on the then current data. At each time step t, a new classifier is generated, called the t^{th} hypothesis h^t (we use the terms *classifier* and *hypothesis* interchangeably). The ensemble obtained by all hypotheses generated up to and including time t, is then referred to as the *composite hypothesis* H^t. With the arrival of each new dataset, Learn^{++}.NSE starts with computing the error E^t of the existing composite hypothesis (H^{t-1}) on the current data in Step 1, which is proportional to the sum of misclassifications of H^{t-1} (2). The normalization factor of $1/m^t$ ensures that $0 \leq E^t \leq 1$ is satisfied. The instance error weights and the penalty distribution are then updated in Step 2 using (3) and (4), respectively. The error weight of instance $x^t(i)$ is reduced by a factor of $E^t < 1$ if it is correctly classified by H^{t-1}. Normalizing the error weights by their sum then provides us with the updated penalty distribution.

Unlike most other ensemble algorithms, the instance error weights (i.e., the penalty distribution) in Learn^{++}.NSE are *not* used for data (re)sampling or instance selection, but rather to weigh and assign error (later in Step 4). In fact, since the environment may change at any time, and that data are received in (possibly small) batches, all training data in are $\mathcal{D}^t$ used for training, returning the hypothesis h^t in Step 3.

In Step 4, the error ε^t of each existing classifier—and not just the most recent h^t—is evaluated on the training data from the current environment. Since classifiers are generated at different times, each receives a different number of evaluations: at time t, h^t gets its first evaluation; whereas h^1 gets its t^{th} evaluation. We use $\varepsilon_k^t, k = 1, \ldots, t$ to denote the error of h_k—the classifier generated at time step k—on dataset (environment) $\mathcal{D}^t$. Henceforth, where applicable, the superscript represents the time index for the current environment, and the subscript is the time the relevant classifier is generated.

Each misclassification does not contribute equally to the error ε_k^t, however, and the penalty distribution D^t is used to weigh these errors: for each misclassified instance i, i.e., when $h_k(x^t(i)) \neq y^t(i)$, the associated penalty weight $D^t(i)$ is added to those of other misclassified instances to obtain the error of classifier h_k on dataset $\mathcal{D}^t$ [(5) in Step 4]. Such an instance error weighting approach ensures that previously misclassified instances are given a higher penalty weight than those correctly classified by the ensemble. More specifically, the relativity of penalties is based on the overall error of the ensemble. When the ensemble does well on the new data—indicating that there has been little or no change in the underlying distributions—misclassified points add higher relative penalty weight (since they should have been learned previously). When the ensemble performs poorly on the new data—indicating that the environment has changed substantially—misclassified data add less relative penalty, since there is little reason to punish unknown instances of a new environment. Hence, classifiers that perform well on novel data are deemed more relevant than others. The goal in this formulation is to allow the ensemble to learn the new knowledge, while reinforcing existing and still relevant knowledge.

If the newest classifier is unable to obtain a weighted error less than 1/2, i.e., if $\varepsilon_{k=t}^t \geq 1/2$, it is discarded since this classifier is not likely to have a positive contribution to the ensemble, and a new classifier is trained in its place. Any other (earlier) classifier, whose error $\varepsilon_{k<t}^t$ is greater than 1/2, has its error saturated at 1/2. When normalized, such that the normalized error β [in (6)] is mapped to [0 1] interval (where 0 represents perfect classification, and 1 represents worst-case classification), an error of $\varepsilon_k^t = 1/2$, is mapped to $\beta_k^t = 1$. A classifier with $\beta_k^t = 1$ receives a final voting weight of zero (9)—but only when evaluated at time t. This process effectively removes classifiers whose performance on the *current* dataset is poor [by assigning a (near) zero voting weight], and is equivalent to forgetting (discarding) the knowledge carried by that classifier. Note, however, the classifier itself is *not* removed, and the forgetting is only temporary. A recurring environment can make an earlier classifier relevant again, triggering a normalized error $\beta_{k<t}^t < 1$, and hence a positive voting weight.

In order to reduce the effects of wide swings in errors, possibly due to outliers or inherent noise in the data, the final voting weight of each classifier is further weighted to emphasize their *recent* performance, using a sigmoidal weighting function (Step 5). The sigmoid-based weights ω_k^t are computed and normalized in (7), using two parameters: parameter a defines the slope and b defines the halfway crossing point of the sigmoid, collectively controlling the number of prior time steps to be considered. These parameters allow averaging classifier decisions over smaller or larger number of time steps, depending on whether the environment is changing slowly or rapidly, respectively. The sigmoidal weights ω_k^t are applied to normalized classifier errors β_k^t to obtain the weighted errors in (8) of Step 5. Fig. 2 illustrates this error weighting mechanism. We emphasize that under this sigmoidal weighting strategy, any classifier containing relevant knowledge about the current environment, *regardless of the classifier's age*, can receive a high voting weight. Classifier age itself has no direct effect on voting weight, but rather it is the classifier's performance on recent environments that determine its "time adjusted" voting weight.

The final voting weights are computed as the logarithm of the reciprocals of the time-adjusted weighted classifier errors in Step 6 (9). The final decision of the ensemble (the composite hypothesis H^t) on an unlabelled data point is then the dynamically weighted [as determined in (9)] majority voting

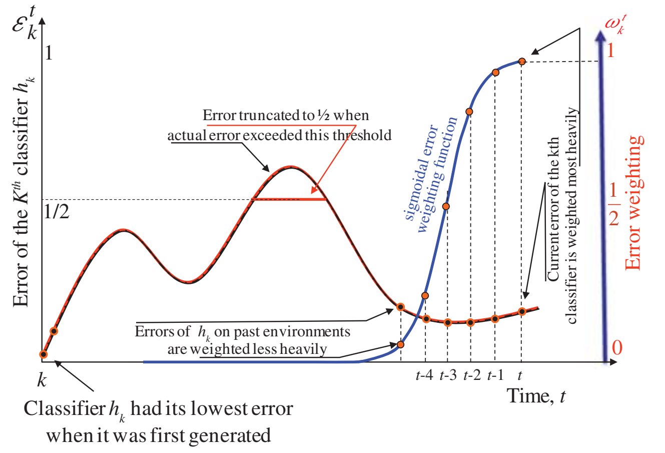
ε_k^t
Error of the K^{th} classifier h_k
1
1/2
Error truncated to ½ when actual error exceeded this threshold
sigmoidal error weighting function
Current error of the kth classifier is weighted most heavily
Errors of h_k on past environments are weighted less heavily
k
Classifier h_k had its lowest error when it was first generated
t-4 t-3 t-2 t-1 t
Time, t
ω_k^t
1
$\frac{1}{2}$
0
Error weighting

Fig. 2. Sigmoidal error weighting in Learn^{++}.NSE.

of all classifiers [(10) of Step 7]. Classifiers with larger voting weights—determined based on their average performance on recent environments—provide the most support for the class chosen by the ensemble.

Two issues are worth addressing before we discuss the experiments. First, since Learn^{++}.NSE continuously adds classifiers, one may be concerned about proliferation of classifiers. This can be addressed by assigning a cap on ensemble size and removing additional classifiers based on their age or error. We do not recommend this, however, as such pruning reduces the ability of the algorithm to remember recurring environments as well as its stability during stationary periods. Our preliminary work showed that performance benefits of retaining the ensemble far outweighs the additional—and modest—computational and memory costs [59]. Second, one may ask whether weaker classifiers should be used to add diversity to the ensemble. We note that the premise of weak-learnability [61] does not apply here, as the fixed distribution assumption is violated in a nonstationary environment. In fact, since such an environment naturally provides diversity, and since the classifiers must learn the new knowledge from limited data in one-pass, we recommend using strong classifiers in the ensemble.

V. Experimental Results

Several datasets simulating different scenarios of nonstationary environments, such as abrupt, gradual, cyclical or variable rate drift, addition or removal of a class, etc. have been generated to determine the behavior of Learn^{++}.NSE, as well as how it compares to other existing approaches. All datasets used in this effort can be downloaded from our site at [62].

The following structure is used in all simulations: experiments begin at $t = 0$ and end at some arbitrary time $t = 1$. Within this interval, T consecutive batches of data are presented for training, where each batch is drawn from a possibly drifting environment, whose rate or nature of drift is assumed unknown. Thus, the number T determines the number of time steps, or snapshots, taken from the data throughout the period of drift. A large T corresponds to a low rate of drift, whereas a small T corresponds to a high effective drift rate, since the algorithm sees fewer snapshots of the data over the same time period. Preliminary results of Learn^{++}.NSE using various effective drift rates (i.e., T values) can be seen in [57]. As one would expect, the ability of the algorithm to track the changing environment is inversely proportional to the rate of drift. Sigmoid parameters were fixed as $a = 0.5$ and $b = 10$. If desired, an active drift detection can be integrated (our future work) to determine these parameters dynamically, though these values worked well on all scenarios we tried.

The Learn^{++}.NSE algorithm is implemented using different base classifiers [NB, SVM, and classification and regression tree (CART)] and compared to other ensemble-based concept drift approaches, such as SEA, DWM, and AdaBoost weighting, which use different learning, weighting and pruning strategies.

SEA is an ensemble-based incremental batch learner employing simple majority voting and classifier pruning (to discard old knowledge) to ensure that the ensemble tracks new environments. Ensemble weights are determined based on classifiers' performance, and weights are also used as the criterion for pruning. The weakest classifier is discarded when the ensemble size exceeds a threshold. Our implementation of SEA is consistent with that in [42], using default ensemble size (25). Both SVM and CART were used as base classifiers in SEA versus Learn^{++}.NSE comparison.

DWM is an online learner that utilizes different weighted majority and ensemble pruning schemes: DWM ensemble is updated periodically only when necessary by adding a classifier or pruning a classifier when its weight drops below a certain performance threshold. DWM pruning is error-based with no upper limit on ensemble size. In this paper, we implement the DWM algorithm using an update period of $\rho = 5$, pruning threshold of $\theta = 0.5$, and a base classifier of NB, consistent with the recommended values in [45]. For a fair comparison, the same NB (and not the stronger SVM) was used with Learn^{++}.NSE.

We also tried Learn^{++}.NSE with an alternative *AdaBoost based weighting* approach as used in *Adaptive Classifier Ensemble-(ACE)* [48] and *Recursive Adaptive Ensemble (REA)* [43], which uses a classifier's most recent error (hence no prior performances considered) in determining voting weight. ACE uses a temporary pruning strategy, with only top-performing classifiers selected for voting, while the rest are ignored, but not discarded. A 95% confidence interval of the top performing classifier is used as the basis, classifiers with weights that lie inside this interval maintain their weights, while others receive a weight of 0 (i.e., they are ignored). Note that this comparison tells us whether the sigmoidal weighting of past errors in Learn^{++}.NSE is beneficial. For brevity, we refer to this weighting scheme as "AdaBoost," though this notation does *not* refer to the Adaboost algorithm itself.

Finally, we also compare Learn^{++}.NSE to a single classifier trained only on the most recent data. Such comparisons are not trivial, as a single classifier trained on the latest data has the best plasticity to track drift, and does not need to be concerned with "classifier baggage." Single classifier comparisons tell us whether using an ensemble of classifiers to weigh in existing knowledge is beneficial in a nonstationary environment.

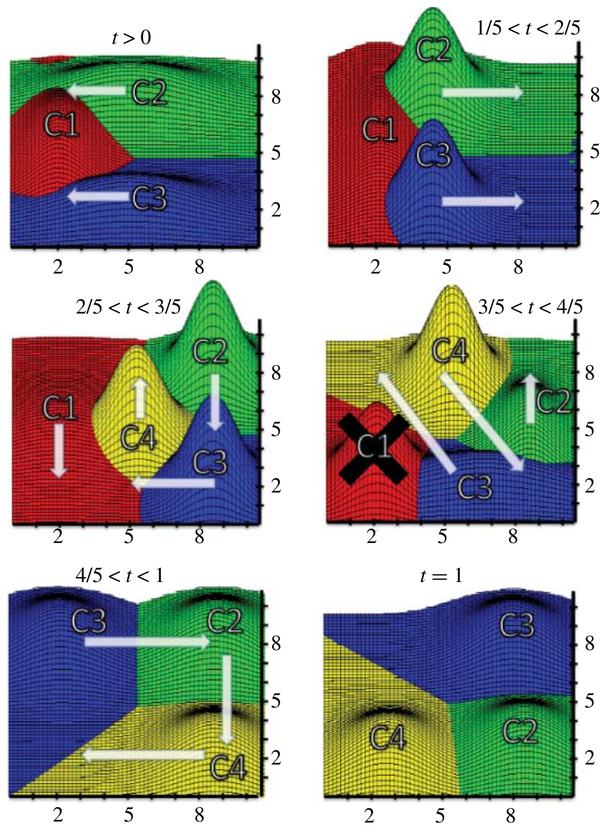

Fig. 3. Gaussian drift data with class addition/removal.

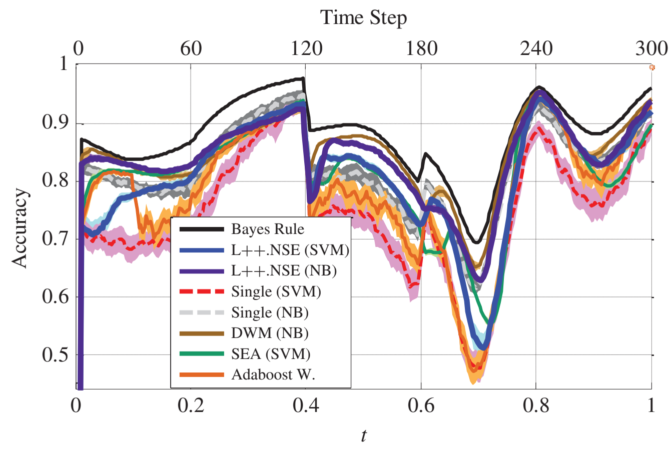

Fig. 4. Comparative results on the Gaussian data.

Essentially, we make three comparisons: comparing batch-based Learn^{++}.NSE to: 1) an online learner DWM, using online learning capable base classifier NB; 2) other batch learners SEA and AdaBoost/ACE using SVMs and CART (strong and weaker learners, respectively); and 3) to a continuously updated single classifier of each of NB, SVM, and CART.

A. Gaussian Drift Data with Class Addition/Removal

This dataset features multiclass data, each drawn from a Gaussian distribution. Such a dataset allows us to control the drift environment, while comparing Learn^{++}.NSE to Bayes classifier. Each class experiences gradual but independent drift, with class means and variances changing according to the parametric equations given in Table I. To make this experiment more challenging, class addition and removal are added to the drifting scenario. Fig. 3 shows six snapshots of the underlying data distributions in the $t = [0 \;\; 1]$ period during which $T = 300$ time steps were seen by the algorithm. At each time step, we select a mere 15–20 samples (only five from each class) to serve as the current training data $\mathcal{D}^t$. At time $t = 2/5$, a new class, C_4 appears (and immediately starts drifting), and class 1, C_1, disappears at time $t = 4/5$. The arrows in Fig. 3 indicate the direction of drift for each class. A movie of the entire scenario is provided in [62]. The results are shown in Fig. 4, where we compare the results of Learn^{++}.NSE using NB and SVM (polynomial kernel, order 6) as base classifiers, to DWM with its default base classifier NB, the SEA algorithm with SVM, Learn^{++}.NSE with AdaBoost- based (non time-averaged) error weighting, the standard Bayes classifier (the best classifier that can be built on this data), as well as single classifiers.

All results in Fig. 4 are averages of 50 independent trials, whose 95% confidence intervals are provided as shading around the performance curves. Each set of results averaged over all times (for this and all other datasets) are summarized in Table II, which also includes Learn^{++}.NSE, SEA, AdaBoost and single classifier using CART as the base classifier (for figure clarity, CART results are not included in the figures).

We make the following observations. First, the Bayes classifier performs best, as expected, followed by algorithms using the NB (also expected due to uncorrelated features). All classifiers see a performance drop at $t = 0.4$, when a new class is added, and a jump at $t = 0.6$, when a class is removed, corresponding to increase and decrease in the complexity of the decision boundaries. There is a major drop in performance in all classifiers at $t = 0.7$, where the most class overlap occurs, representing the most difficult classification problem.

DWM and Learn^{++}.NSE (with NB) are best performers (with no significant difference between the two); both providing quick recovery for abrupt changes thanks to online nature of DWM, and the specific weighting strategy of Learn^{++}.NSE.

Among batch learners, Learn^{++}.NSE outperformed both AdaBoost (with significance) and SEA (both with SVM and CART), despite this dataset, not including any recurring environment, favoring the other two algorithms. These results indicate that Learn^{++}.NSE can consistently employ past classifiers recognizing those parts of the feature space previously seen. We believe AdaBoost type weighting suffers from non-optimal weighting (discarding past performance) especially as the ensemble grows, and SEA suffers from slow reaction to change due to uniform voting. Finally, we observe that Learn^{++}.NSE always outperformed–with significance–a single classifier trained on the same (NB, SVM or CART) base classifier (see Table II), indicating that past knowledge is indeed being successfully utilized by Learn^{++}.NSE.

TABLE I
PARAMETRIC EQUATIONS FOR GAUSSIAN DRIFT DATA

	$0 < t < 1/5$				$1/5 < t < 2/5$			
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y
C1	2	5	1	2 + 5t	2	5	1 + 5t	3 − 5t
C2	5 − 5t	8	3 − 10t	1	4 + 20t	8	1	1
C3	5 − 5t	2	3 − 10t	1	4 + 20t	2	1	1
C4	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	$2/5 < t < 3/5$				$3/5 < t < 4/5$			
C1	2	5 − 15t	2 − 5t	2 − 5t	N/A	N/A	N/A	N/A
C2	8	8 − 20t	1	1 + 5t	8	4 + 20t	1 + 2.5t	2 − 2.5t
C3	8 − 10t	2	1 + 10t	1	6 − 20t	2 + 30t	3 − 7.5t	1 + 2.5t
C4	5	5 + 15t	1	1	5 + 15t	8 − 30t	1 + 2.5t	1 + 2.5t

	$4/5 < t < 1$			
	μ_x	μ_y	σ_x	σ_y
C1	N/A	N/A	N/A	N/A
C2	8	8 − 30t	1.5	1.5
C3	2 + 30t	2	1.5	1.5
C4	8 − 30t	2	1.5	1.5

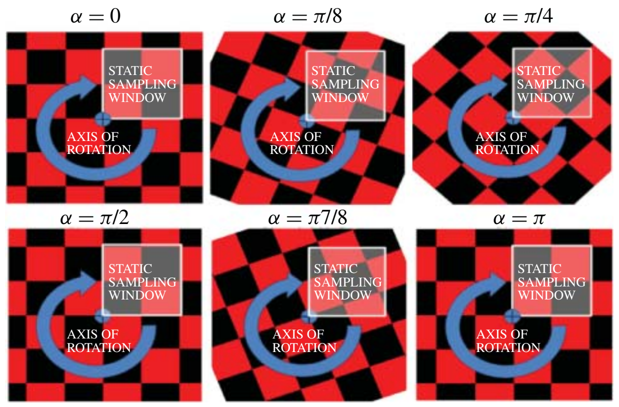

Fig. 5. Snapshots from a $^1/_2$ rotation of the checkerboard data.

B. Rotating Checkerboard Dataset: Variable Rate Drift

A non-Gaussian data set is derived from the canonical XOR problem, which resembles a rotating checkerboard. As shown in Fig. 5, the rotation makes this deceptively simple-looking problem particularly challenging, as the angle and location of the decision boundaries change drastically every few time steps. Fig. 5 shows half a rotation ($\alpha = 0$ to π), indexed to the parameter α, where the axis of rotation is the lower left corner of the sampling window. After half a rotation, data are drawn from a recurring environment, as the $[\pi\ \ 2\pi]$ interval creates an identical distribution drift to that of the $[0\ \ \pi]$ interval. In order to prevent training on identical snapshots of data and to increase complexity, 10% random noise was introduced. Each training dataset is kept particularly small, consisting of a mere 25 samples (total from both classes) drawn from the sampling window, making this data further challenging to the learner. Test data are composed of 1024 data points uniformly sampled from the current distribution's entire grid at a 32-by-32 resolution, sufficient enough to evaluate the learner's ability to approximate the sharp angles of the true decision boundary.

Perhaps the most challenging and unique aspect of this experiment, however, is the variability introduced in the drift rate. Fig. 6 shows the four drift rate scenarios designed to determine the algorithms' behavior under variable rate drift. All using 400 time steps from $t = 0$ to $t = 1$, these are: 1) constant drift rate of 2pi/400 = 0.016rad/time step; 2) exponentially increasing drift rate (the board rotates increasingly faster); 3) sinusoidally varying drift rate; and 4) Gaussian pulse shaped (slow–fast–very fast–fast–slow) drift rate. The data and movie files describing these scenarios can be found at [62].

Fig. 7 shows result of 50 independent trials on test data (entire grid as shown in Fig. 5). Each performance curve is enclosed by its 95% confidence interval to determine statistical significance of the performance differences. Generalization performances averaged across all time steps are also provided in Table II. We make the following observations. First, Learn^{++}. NSE, using the strong learner SVM, outperforms all other algorithms, including the single SVM classifier or other SVM-based ensemble approaches, usually with wide significance. Second, as expected, algorithms that use NB as base classifier perform poorly, due to the nature of these data whose features are class conditionally correlated. Third, when the environment is changing slowly, for example, during early and late sections of pulse drift [Fig. 7(b)] and mid sections of the sinusoidal drift [Fig. 7(d)], the ensemble performances increase rapidly, and better track the environment compared to when the rate of change is accelerating (e.g., $t = 0.2 - 0.5$ s on pulse drift, $t = 0.7 - 1$ s on exponential drift). We should mention that the sharp performance peaks in all Fig. 7 plots are simply due to the periodic nature of the problem, with decision boundaries becoming perpendicular (and hence simpler) for every $\pi/2$ radians. Of course, the number of time steps the board takes to reach multiples of $\pi/2$ radians varies

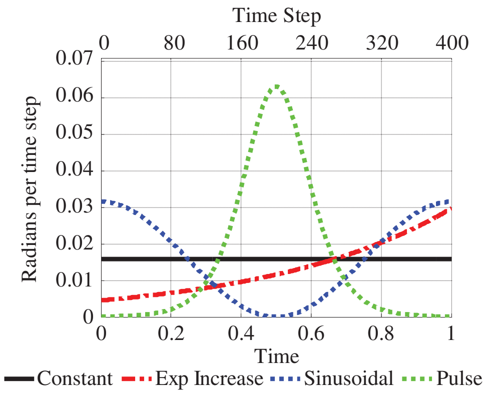

Fig. 6. Variable drift rate controlled by the rate at which α parameter is updated for rotating checkerboard dataset.

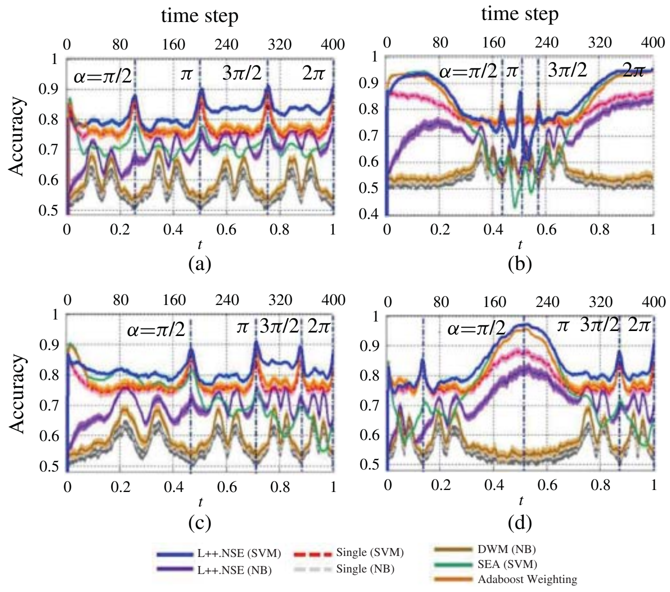

Fig. 7. Performances on checkerboard data with (a) constant, (b) gaussian pulse, (c) exponential, and (d) sinusoidal drift rate.

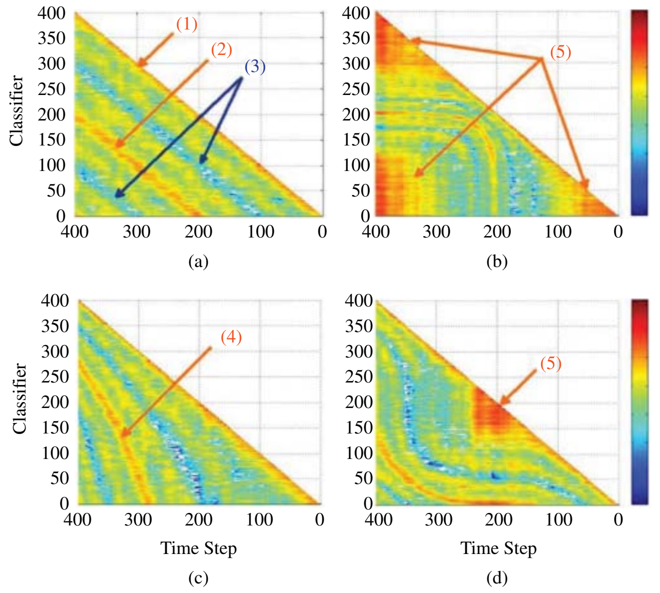

Fig. 8. Weight distribution (max in red, min in blue) over time for checkerboard dataset with (a) constant, (b) pulsing, (c) exponential, and (d) sinusoidal drift rate.

according to drift scenario: for constant drift [Fig. 7(a)], they are at t = [0 100 200 300 400]; for pulse drift Fig. 7(b)], at t = [0 180 200 220 400], etc.

One of the more interesting observations is the behavior of the algorithm after $\alpha = \pi$, This happens at $t = 200, 200, 275$, and 200 for the constant, pulse, exponential and sinusoidal drifts, respectively, after which the distribution repeats itself in the $\alpha = [\pi \sim 2\pi]$ interval, creating a cyclic environment. Learn^{++}.NSE shows a significant increase in performance after this interval, compared to $\alpha = [0 \sim \pi]$ interval, indicating the ability of the algorithm to make effective use of its prior knowledge, by reactivating early classifiers during the recurring environments. This is particularly striking in Fig. 7(a), and even more so in Fig. 7(c), where the performance improvement due to reactivating old classifiers outweighs the performance drop due to rapidly accelerating rate of drift that also occurs at around $t = 275(\alpha = \pi)$. The second half performances in Fig. 7(b) and (d) are also higher than those of the first half. Among all comparisons, only Learn^{++}.NSE showed such an improvement. DWM is limited by the use of an online classifier that cannot sufficiently update to learn the complex decision boundary, SEA loses all prior knowledge and thus shows no improvement over the recurring environment, and Adaboost weighting is inconsistent in assigning appropriate weight to the most relevant knowledge at a given time step even in the presence of recurring data.

In summary, we note that in all base-classifier matched comparisons, Learn^{++}.NSE provides the best performance: in online learner comparison, Learn^{++}.NSE with NB significantly outperforms DWM with NB, as well as single continuously updated NB; among batch learners, Learn^{++}.NSE outperforms SEA and AdaBoost, either with SVM or CART (see Table II for CART results), and Learn^{++}.NSE ensemble always outperforms, with significance, any single classifier trained on the same base classifier (NB, SVM or CART). Furthermore, these results apply regardless of the type of drift scenario.

Perhaps a more dramatic proof of Learn^{++}.NSE's ability to reactivate old classifiers, precisely when they would be most beneficial, can be seen in Fig. 8, which provides pseudo color images of member classifier weights at each time step after their creation. The main diagonal [Arrow (1)] represents the average weight of each classifier at the time it is created (averaged over 50 trials), a vertical cross-section at some $t = t^*$ shows the average weights of all classifiers at time t^*, whereas a horizontal cross-section for any classifier (starting at the diagonal and moving left) indicates the average weights of that classifier since its creation at each subsequent time step. The off-diagonal or curved patterns (starting at some time step t^{**} on the horizontal axis and moving toward the vertical axis) indicate the average weight of all classifiers t^{**} steps after their creation. For example, Fig. 8(a) shows that each classifier receives a very high weight, (i.e., it gets reactivated), exactly 200 steps after its creation [the red off-

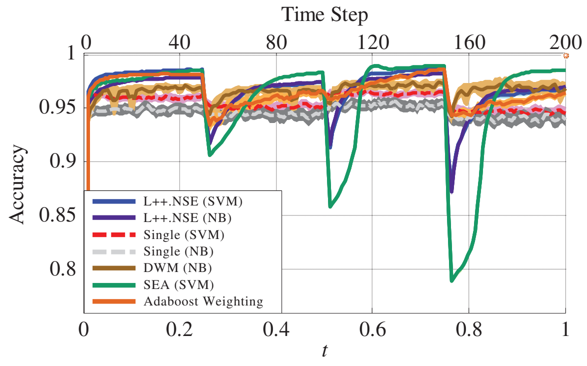

Fig. 9. Comparative performances on SEA dataset (SVM polynomial kernel, order: 2).

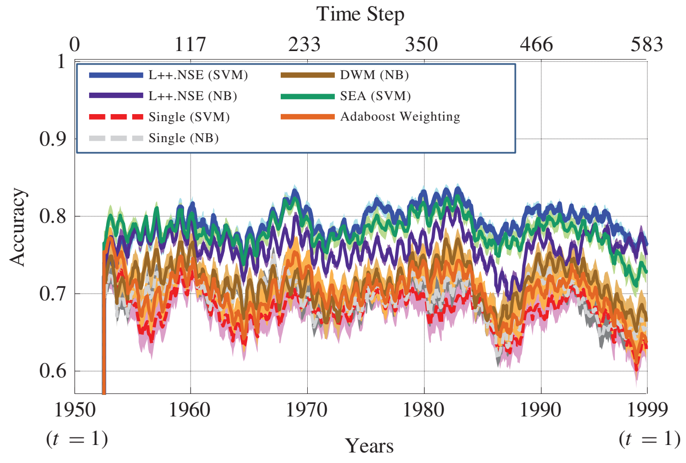

Fig. 10. Comparative performances on weather dataset (SVM polynomial kernel, order: 2).

diagonal, starting at $t = 200$, Arrow (2)]. This is a very satisfying observation, as in the constant drift rate experiment, each classifier experiences a recurring environment exactly 200 steps after its creation. The (blue) off-diagonals (Arrow 3) starting at steps 100 and 300 show reduced weights (i.e., classifiers are de-activated), precisely when the checkerboard pattern is reversed (class definitions flip), and during which we would expect each classifier to be least useful. The curved patterns in Fig. 8(b)–(d) shows similar behavior, with the curves indicating the precise time-varying nature of the drifts. For example, the red curve (Arrow 4) starting at $t = 275$ in Fig. 8(c) indicates that the first classifier (created at $t = 0$) waits 275 time steps to be reactivated; whereas classifiers generated later are activated increasingly sooner: e.g., classifier created at $t = 200$ gets reactivated at time step 350. This makes sense, as in this experiment the board rotates (and the environment recurs) increasingly faster with each time step and hence later classifiers get reactivated faster compared to earlier classifiers. Another interesting observation is the high weights subsequent classifiers receive when the environment is near stationary (Arrow 5). Similar patterns can be seen in other figures where the weight distributions closely follow the change in drift rate which directly controls how quickly the distributions repeat themselves.

C. *SEA Concepts*

The SEA Concepts is developed by Street [42] and has been used by several algorithms as a standard test for concept change. This is the dataset on which SEA algorithm was originally tested. The dataset is characterized by extended periods without any drift with occasional sharp changes in the class boundary, i.e., sudden drift or *concept change*. The dataset includes two classes and three features, with only two features being relevant, and the third being noise. Class labels are assigned based on the sum of the relevant features, and are differentiated by comparing this sum to a threshold that separates a 2-D hyper-plane: an instance is assigned to class 1 if the sum of its (relevant) features $(f_1 + f_2)$ fall below the threshold, and assigned to class 2, otherwise. At regular intervals, the threshold is changed, creating an abrupt shift in the class boundary. Data are uniformly distributed between 0 and 10, and the threshold θ_t is changed three times throughout the experiment with increasing severity (8→9→7.5→9.5). Training procedure is identical to that described in [42]: 50 000 points are introduced as training data (25 000 points per class), in 200 time steps, 250 points/time step. Also as per [42], 10% class noise is added to the training data. A separate set of 50 000 data points from each environment (with no noise) are used for testing. The results are shown in Fig. 9 and Table II.

We make the following observations from these results. DWM has the best recovery rate after concept change in comparison with Learn^{++}.NSE (with NB), yet a relatively low convergence in stationary environments, presumably due to throwing out recent and relevant classifiers during slow or no change periods. Conversely, the batch-learning Learn^{++}.NSE provides a higher convergence (final performance) but with a slower (than DWM) recovery rate. These results point to a tradeoff between recovery rate and convergence. Among batch classifiers, SEA has the best convergence in a stationary environment, yet very poor recovery after concept change even on its own benchmark dataset. Adaboost weighting also provides slow recovery and poor convergence after concept drift, indicating that too much weight is being assigned to old classifiers. Learn^{++}.NSE (both with NB and SVM), provides a very good balance between recovery (plasticity) during rapid changes, and the ability to reach and sustain high performance (stability) during slow or no drift scenarios. We believe that this is due to its unique error weighting strategy. Note that single classifiers do not experience any dip in the performance as they do not have any baggage; however, they are also unable to match the performance of ensemble approaches. We should add that, when averaged across time, Learn^{++}.NSE does outperform all other base-classifier matched algorithms, and with significance in most cases (Table II).

D. *Nebraska Weather Prediction Data*

The U.S. National Oceanic and Atmospheric Administration has compiled weather measurements from over 9000 weather stations worldwide [63]. Records date back to the 1930s, providing a wide scope of weather trends. Daily measurements include a variety of features (temperature, pressure,

TABLE II
Time Averaged Performance Comparisons

		Gaussian	SEA	Weather	
	Bayes	88.1 +/– 0.0			
L++.NSE versus Online	L++.NSE (NB)	84.0 +/– 0.5	96.6 +/– 0.2	75.9 +/– 0.7	L++.NSE versus base-classifier matched single classifier
	DWM (NB)	84.8 +/– 0.4	96.6 +/– 0.6	71.3 +/– 1.8	
	Single (NB)	82.3 +/– 1.2	94.7 +/– 0.6	69.4 +/– 1.4	
L++.NSE versus Other Batch Alg.	L++.NSE (SVM)	81.0 +/– 0.9	96.8 +/– 0.2	78.8 +/– 1.0	
	SEA (SVM)	81.3 +/– 0.5	95.7 +/– 0.2	77.8 +/– 1.1	
	Adaboost (SVM)	78.6 +/– 1.7	96.6 +/– 0.3	70.2 +/– 1.9	
	Single (SVM)	74.6 +/– 2.4	95.6 +/– 0.4	67.8 +/– 2.0	
	L++.NSE (CART)	82.8 +/– 0.7	95.8 +/– 0.5	75.7 +/– 1.1	
	SEA (CART)	81.7 +/– 0.5	95.6 +/– 0.3	72.8 +/– 1.0	
	Adaboost (CART)	81.3 +/– 1.3	87.8 +/– 0.9	68.5 +/– 1.9	
	Single (CART)	77.7 +/– 1.9	86.7 +/– 1.0	66.8 +/– 2.0	

		CB (constant)	CB (pulse)	CB (exp)	CB (sinusoid)	
L++.NSE Online	L++.NSE (NB)	69.9 +/– 1.3	70.5 +/– 1.6	69.1 +/– 1.4	71.1 +/– 1.5	L++.NSE versus base-classifier matched single classifier
	Single (NB)	56.6 +/– 1.7	54.3 +/– 1.7	56.5 +/– 1.7	55.3 +/– 1.7	
	DWM (NB)	59.6 +/– 1.6	56.4 +/– 1.7	59.6 +/– 1.7	57.9 +/– 1.7	
L++.NSE versus Other Batch Alg.	L++.NSE (SVM)	81.9 +/– 0.9	84.0 +/– 0.7	81.6 +/– 0.9	83.5 +/– 0.9	
	SEA (SVM)	71.6 +/– 0.7	78.5 +/– 0.6	73.0 +/– 0.7	75.4 +/– 0.6	
	Adaboost (SVM)	77.8 +/– 1.7	83.7 +/– 1.1	78.0 +/– 1.4	80.9 +/– 1.2	
	Single (SVM)	76.6 +/– 1.5	79.9 +/– 1.5	76.6 +/– 1.5	78.6 +/– 1.5	
	L++.NSE (CART)	77.3 +/– 1.1	81.2 +/– 1.0	77.0 +/– 1.1	79.4 +/– 1.0	
	SEA (CART)	69.3 +/– 0.9	77.2 +/– 0.8	70.6 +/– 0.9	73.2 +/– 0.8	
	Adaboost (CART	71.6 +/– 1.7	80.1 +/– 1.4	72.2 +/– 1.6	75.9 +/– 1.6	
	Single (CART)	67.8 +/– 1.9	69.3 +/– 2.6	67.7 +/– 1.9	68.7 +/– 2.3	

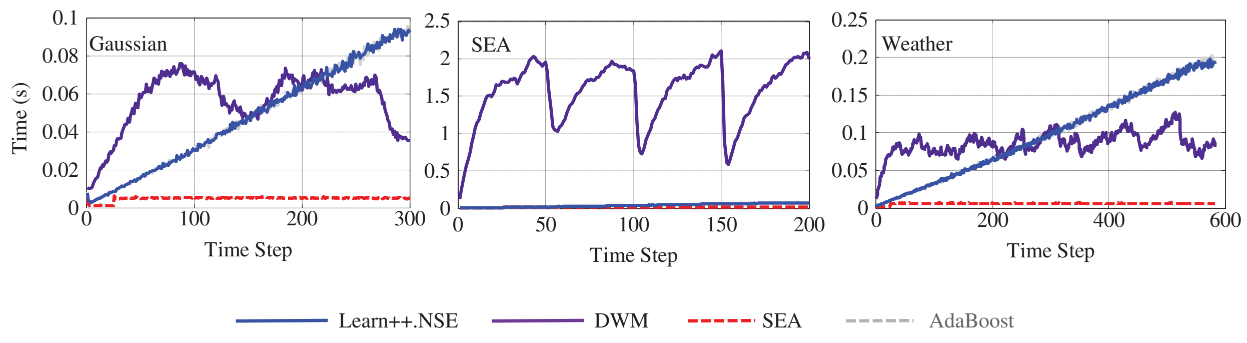

Fig. 11. Timing diagrams for Gaussian, SEA and weather datasets (with NB classifier).

wind speed, etc.) and indicators for precipitation and other weather-related events. As a meaningful real world dataset, we chose the Offutt Air Force Base in Bellevue, Nebraska, for this experiment due to its extensive range of 50 years (1949–1999) and diverse weather patterns, making it a long-term precipitation classification/prediction drift problem.

Eight features were selected based on their availability, eliminating those with a missing feature rate above 15%. The remaining missing values were imputed by the mean of features in the preceding and following instances. Class labels are based on the binary indicator(s) provided for each daily reading of *rain* with 18 159 daily readings: 5698 (31%) positive (rain) and 12 461 (69%) negative (no rain). Each training batch consisted of 30 samples (days), with corresponding test data selected as the *subsequent* 30 days. Thus, the learner is asked to *predict* the next 30 days' forecast, which becomes the training data in the next batch. The dataset included 583 consecutive "30-day" time steps covering 50 years.

Fig. 10 and Table II show the comparative results, from which we make following observations: Learn^{++}.NSE outperforms all other algorithms regardless of the base classifier being used, with SVM providing the best performance; all pair wise differences, i.e., Learn^{++}.NSE versus DWM with NB, Learn^{++}.NSE versus SEA, Adaboost Weighting or single classifier with SVM or CART (see Table II) are statistically significant at all time steps with wide margins, except with SEA where the difference is significant only at certain time instances. Single classifiers, with SVM, NB or CART, performed the worst.

We also observe a strong sinusoidal component in ensemble performances; computing the Fourier transform revealed a very strong spectral component corresponding to exactly 1 year, demonstrating the cyclical drift inherent in the data.

Finally, comparing Learn^{++}.NSE to itself on different base classifiers, we observe that SVM outperforms CART as well as NB (except when features are uncorrelated, when NB becomes strong learner), confirming our belief that strong classifiers should be preferred in concept drift problems.

We conclude this section with a brief discussion on computational cost of the algorithms under study. Although this paper is not specific to data-streams, and time and memory consumption are not primary concerns, such discussion provides some insight into the algorithms' behavior. Clearly, computational efficiency is not a strong attribute of ensemble-based approaches: by its very nature, when using an ensemble system instead of a single classifier, we accept a higher computational cost in return for qualities not possible with a single classifier, e.g., ability to handle recurrent environments and class addition or removal. Nevertheless, the complexity of Learn^{++}.NSE is only linear in the number of classifiers, and since one classifier is generated per dataset, also linear in the number of data batches. The actual complexity of the algorithm depends on the complexity of the base model used (e.g., SVM is costlier than NB). For SEA, computational order is constant after reaching the max threshold, whereas that of DWM depends on update rate, and pruning threshold.

Fig. 11 shows three examples of the average *learning time per time step* for all algorithms. Learning time includes the time to train new classifiers *and* re-weigh the ensemble members. The learning time for Learn^{++}.NSE increase linearly since all classifiers are maintained *and* reweighted on the most recent training data. SEA maintains a steady learning time once the ensemble reaches a maximum size and then pruned. The learning time for DWM is more unpredictable, since the ensemble size is dynamic and classifiers may be added or pruned, with no upper threshold for ensemble size. DWM's *per-time-step* computational time increases for large datasets, particularly when old classifiers are not removed during slow periods, as in the SEA data, for which DWM takes the longest time (far exceeding that of other algorithms on this dataset).

TABLE III
ALGORITHM RUNTIME SUMMARY

Time Steps:	300	200	583	400
Samples (Train/Test):	20/1024	250/250	30/30	25/1024
Learn^{++}.NSE (s):	50.32	17.21	117.08	71.35
SEA (s):	10.41	5.96	12.26	12.97
DWM (s):	21.11	317.54	53.84	33.17
Adaboost (s):	82.88	20.59	135.83	115.24

Whereas batch learners are capable of evaluating and training on large amounts of new data at once, DWM must do so on an instance-by-instance basis, a costly approach as the training size increases. Table III provides total runtime averaged over 50 trials, from start ($t = 0$) to finish ($t = 1$). As expected, SEA runs the fastest due to its fixed ensemble size, DWM runtime depends on training size, and the Learn^{++}.NSE and AdaBoost weighting depends on the number of time steps. Memory wise, SEA is again the most frugal algorithm, due to fixed ensemble size, followed by DWM, and Learn^{++}.NSE/AdaBoost. All experiments were run on Intel Core i7 CPU at 2.67GHz.

VI. CONCLUSION

We described an ensemble of classifiers-based approach, Learn^{++}.NSE, for learning in nonstationary environments. The novelty of Learn^{++}.NSE is its strategic use of current and past classifiers combined with dynamically updated voting weights, based on their time adjusted errors on current and recent environments. Such a weighting mechanism allows Learn^{++}.NSE to learn new knowledge by creating new classifiers, while using existing knowledge when such knowledge is still relevant. The primary contribution of Learn^{++}.NSE is therefore its versatility as a general framework for learning in nonstationary environments. While many algorithms perform well on a particular type of drifting environment (e.g., SEA performs well on concept change, DWM performs well on relatively gradual drifts, etc.), Learn^{++}.NSE can accommodate a wide variety of drift scenarios, regardless of whether it is gradual, abrupt, slow, fast or cyclical, or even variable rate drift—the last two of which are not generally addressed by other approaches. The weight analysis of the algorithm demonstrates that the unique weight assigning strategy used by Learn^{++}.NSE makes very efficient use of existing knowledge by reactivating early classifiers precisely when they are needed the most, and by temporarily disabling them when they are not relevant. This mechanism allows the algorithm to learn new knowledge, temporarily forget irrelevant knowledge, and then recall such knowledge when it becomes relevant again. To the best of our knowledge, Learn^{++}.NSE is the only algorithm that has this unique capability. Experiments also supported our assertion that strong learners are desired for non-stationary datasets (e.g., NB for data with class conditionally independent features, and SVM for other general data). We have also shown that the learning mechanisms used by Learn^{++}.NSE are consistent with that of human learning according to two well-established human learning theories, schema and scaffolding. Our future work will focus on the statistical analysis of Learn^{++}.NSE for possible performance guarantees on different NSE scenarios.

Acknowledgement

The authors would like to acknowledge M. Muhlbaier and M. Karnick for their contributions to early stages of this paper, and J. Kounios for discussions on Human Learning.

References

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

[2] F. C. Bartlett, *Remembering: A Study in Experimental and Social Psychology*. Cambridge, U.K.: Cambridge Univ. Press, 1932.

[3] J. H. Flavell, "Piaget's legacy," *Psychol. Sci.*, vol. 7, no. 4, pp. 200–203, Jul. 1996.

[4] L. S. Vygotsky, *Mind and Society: The Development of Higher Psychological Processes*. Cambridge, U.K.: Harvard Univ. Press, 1978.

[5] J. Piaget, *Six Psychological Studies*. New York: Random House, 1967.

[6] M. H. Appel and L. S. Goldberg, *Equilibration: Theory, Research, and Application*. New York: Plenum, 1977.

[7] B. J. Reiser, "Scaffolding complex learning: The mechanisms of structuring and problematizing student work," *J. Learn. Sci.*, vol. 13, no. 3, pp. 273–304, 2004.

[8] D. Wood, "Scaffolding, contingent tutoring and computer-based learning," *Int. J. Artif. Intell. Educ.*, vol. 12, no. 3, pp. 280–292, 2001.

[9] A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen, "Handling local concept drift with dynamic integration of classifiers: Domain of antibiotic resistance in nosocomial infections," in *Proc. 19th IEEE Int. Symp. Comput.-Based Med. Syst.*, Salt Lake City, UT, Jul. 2006, pp. 679–684.

[10] J. Gao, W. Fan, J. Han, and P. S. Yu, "A general framework for mining concept-drifting data streams with skewed distributions," in *Proc. SIAM Int. Conf. Data Min.*, vol. 7. 2007, pp. 3–14.

[11] J. Quinonero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset Shift in Machine Learning*. Cambridge, MA: MIT Press, 2009.

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

[13] L. L. Minku, A. P. White, and Y. Xin, "The impact of diversity on online ensemble learning in the presence of concept drift," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 5, pp. 730–742, May 2010.

[14] L. I. Kuncheva, "Classifier ensembles for detecting concept change in streaming data: Overview and perspectives," in *Proc. Eur. Conf. Artif. Intell.*, 2008, pp. 5–10.

[15] L. I. Kuncheva, "Classifier ensembles for changing environments," in *Multiple Classifier Systems*, vol. 3077. New York: Springer-Verlag, 2004, pp. 1–15.

[16] R. Polikar, L. Udpa, S. S. Udpa, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Trans. Syst., Man Cybern. Part C: Appl. Rev.*, vol. 31, no. 4, pp. 497–508, Nov. 2001.

[17] M. Muhlbaier, A. Topalis, and R. Polikar, "Learn++.NC: Combining ensemble of classifiers with dynamically weighted consult-and-vote for efficient incremental learning of new classes," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 152–168, Jan. 2009.

[18] J. C. Schlimmer and R. H. Granger, "Incremental learning from noisy data," *Mach. Learn.*, vol. 1, no. 3, pp. 317–354, 1986.

[19] R. Klinkenberg, "Learning drifting concepts: Example selection versus example weighting," *Intell. Data Anal.*, vol. 8, no. 3, pp. 281–300, Aug. 2004.

[20] M. Nunez, R. Fidalgo, and R. Morales, "Learning in environments with unknown dynamics: Toward more robust concept learners," *J. Mach. Learn. Res.*, vol. 8, pp. 2595–2628, Nov. 2007.

[21] P. Wang, H. Wang, X. Wu, W. Wang, and B. Shi, "A low-granularity classifier for data streams with concept drifts and biased class distribution," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 9, pp. 1202–1213, Sep. 2007.

[22] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence* (Lecture Notes in Computer Science), vol. 3171. New York: Springer-Verlag, 2004, pp. 286–295.

[23] C. Alippi and M. Roveri, "Just-in-time adaptive classifiers—Part I: Detecting nonstationary changes," *IEEE Trans. Neural Netw.*, vol. 19, no. 7, pp. 1145–1153, Jul. 2008.

[24] C. Alippi and M. Roveri, "Just-in-time adaptive classifiers—Part II: Designing the classifier," *IEEE Trans. Neural Netw.*, vol. 19, no. 12, pp. 2053–2064, Dec. 2008.

[25] L. Cohen, G. Avrahami-Bakish, M. Last, A. Kandel, and O. Kipersztok, "Real-time data mining of non-stationary data streams from sensor networks," *Inf. Fus.*, vol. 9, no. 3, pp. 344–353, Jul. 2008.

[26] M. Markou and S. Singh, "Novelty detection: A review—Part 2: Neural network based approaches," *Signal Process.*, vol. 83, no. 12, pp. 2499–2521, Dec. 2003.

[27] L. Rutkowski, "Adaptive probabilistic neural networks for pattern classification in time-varying environment," *IEEE Trans. Neural Netw.*, vol. 15, no. 4, pp. 811–827, Jul. 2004.

[28] E. A. de Oliveira, "The Rosenblatt Bayesian algorithm learning in a nonstationary environment," *IEEE Trans. Neural Netw.*, vol. 18, no. 2, pp. 584–588, Mar. 2007.

[29] N. G. Pavlidis, D. K. Tasoulis, N. M. Adams, and D. J. Hand, "λ-perceptron: An adaptive classifier for data streams," *Pattern Recognit.*, vol. 44, no. 1, pp. 78–96, Jan. 2011.

[30] P. Vorburger and A. Bernstein, "Entropy-based concept shift detection," in *Proc. 6th Int. Conf. Data Min.*, 2006, pp. 1113–1118.

[31] S. Hoeglinger and R. Pears, "Use of Hoeffding trees in concept based data stream mining," in *Proc. Int. Conf. Inf. Autom. Sustain.*, Melbourne, Australia, Dec. 2007, pp. 57–62.

[32] C.-J. Tsai, C.-I. Lee, and W.-P. Yang, "Mining decision rules on data streams in the presence of concept drifts," *Expert Syst. Appl.*, vol. 36, no. 2, pp. 1164–1178, Mar. 2009.

[33] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proc. Conf. Knowl. Disc. Data*, 2001, pp. 97–106.

[34] L. Cohen, G. Avrahami, M. Last, and A. Kandel, "Info-fuzzy algorithms for mining dynamic data streams," *Appl. Soft Comput.*, vol. 8, no. 4, pp. 1283–1294, Sep. 2008.

[35] S. Srinivasan, J. Samuelsson, and W. B. Kleijn, "Codebook-based Bayesian speech enhancement for nonstationary environments," *IEEE Trans. Audio, Speech Lang. Process.*, vol. 15, no. 2, pp. 441–452, Feb. 2007.

[36] D. R. Lowne, S. J. Roberts, and R. Garnett, "Sequential non-stationary dynamic classification with sparse feedback," *Pattern Recognit.*, vol. 43, no. 3, pp. 897–905, Mar. 2010.

[37] A. Blum, "Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain," *Mach. Learn.*, vol. 26, no. 1, pp. 5–23, Jan. 1997.

[38] Z. Xingquan, W. Xindong, and Y. Ying, "Dynamic classifier selection for effective mining from noisy data streams," in *Proc. 4th IEEE Int. Conf. Data Min.*, Nov. 2004, pp. 305–312.

[39] N. Littlestone, "Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm," *Mach. Learn.*, vol. 2, no. 4, pp. 285–318, Apr. 1988.

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

[41] N. Oza, "Online ensemble learning," Ph.D. dissertation, Dept. Comput. Sci., Univ. California, Berkeley, 2001.

[42] W. N. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," in *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2001, pp. 377–382.

[43] S. Chen and H. He, "Toward incremental learning of nonstationary imbalanced data stream: A multiple selectively recursive approach," *Evolv. Syst.*, vol. 2, no. 1, pp. 35–50, 2011.

[44] A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen, "Dynamic integration of classifiers for handling concept drift," *Inf. Fus.*, vol. 9, no. 1, pp. 56–68, Jan. 2008.

[45] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *J. Mach. Learn. Res.*, vol. 8, pp. 2755–2790, Dec. 2007.

[46] J. Gao, W. Fan, and J. Han, "On appropriate assumptions to mine data streams: Analysis and practice," in *Proc. Int. Conf. Data Min.*, 2007, pp. 143–152.

[47] H. Wang, W. Fan, P. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proc. ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2003, pp. 226–235.

[48] K. Nishida and K. Yamauchi, "Adaptive classifiers-ensemble system for tracking concept drift," in *Proc. Int. Conf. Mach. Learn. Cybern.*, vol. 6. Hong Kong, Aug. 2007, pp. 3607–3612.

[49] H. He and S. Chen, "IMORL: Incremental multiple-object recognition and localization," *IEEE Trans. Neural Netw.*, vol. 19, no. 10, pp. 1727–1738, Oct. 2008.

[50] J. Gao, B. Ding, F. Wei, H. Jiawei, and P. S. Yu, "Classifying data streams with skewed class distributions and concept drifts," *IEEE Internet Comput.*, vol. 12, no. 6, pp. 37–49, Nov.–Dec. 2008.

[51] H. Abdulsalam, D. B. Skillicorn, and P. Martin, "Classification using streaming random forests," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 1, pp. 22–36, Jan. 2011.

[52] A. Bifet, "Adaptive learning and mining for data streams and frequent patterns," Ph.D. dissertation, Dept. Lleng. Sist. Inf., Univ. Politècnica Catalunya, Barcelona, Spain, Apr. 2009.

[53] A. Bifet, E. Frank, G. Holmes, and B. Pfahringer, "Accurate ensembles for data streams: Combining restricted Hoeffding trees using stacking," in *Proc. 2nd Asian Conf. Mach. Learn.*, vol. 13. 2010, pp. 1–16.

[54] A. Bifet. (2010, Dec. 30). *MOA: Massive Online Analysis* [Online]. Available: http://moa.cs.waikato.ac.nz

[55] M. Scholz and R. Klinkenberg, "Boosting classifiers for drifting concepts," *Intell. Data Anal.*, vol. 11, no. 1, pp. 3–28, Jan. 2007.

[56] R. Polikar, J. DePasquale, H. S. Mohammed, G. Brown, and L. I. Kuncheva, "Learn^{++}.MF: A random subspace approach for the missing feature problem," *Pattern Recognit.*, vol. 43, no. 11, pp. 3817–3832, Nov. 2010.

[57] M. Karnick, M. Ahiskali, M. D. Muhlbaier, and R. Polikar, "Learning concept drift in nonstationary environments using an ensemble of classifiers based approach," in *Proc. Int. Joint Conf. Neural Netw.*, Hong Kong, 2008, pp. 3455–3462.

[58] M. Karnick, M. D. Muhlbaier, and R. Polikar, "Incremental learning in non-stationary environments with concept drift using a multiple classifier based approach," in *Proc. 19th Int. Conf. Pattern Recognit.*, Tampa, FL, Dec. 2008, pp. 1–4.

[59] R. Elwell and R. Polikar, "Incremental learning in nonstationary environments with controlled forgetting," in *Proc. Int. Joint Conf. Neural Netw.*, Atlanta, GA, Jun. 2009, pp. 771–778.

[60] R. Elwell and R. Polikar, "Incremental learning of variable rate concept drift," in *Proc. Int. Workshop Multiple Class. Syst.*, vol. 5519. 2009, pp. 142–151.

[61] R. E. Schapire, "The strength of weak learnability," *Mach. Learn.*, vol. 5, no. 2, pp. 197–227, Jun. 1990.

[62] R. Polikar and R. Elwell. (2011, Jun. 18). *Benchmark Datasets for Evaluating Concept Drift/NSE Algorithms* [Online]. Available: http://users.rowan.edu/~polikar/research/NSE

[63] U.S. National Oceanic and Atmospheric Administration. *Federal Climate Complex Global Surface Summary of Day Data* [Online]. Available FTP: ftp.ncdc.noaa.gov/pub/data/gsod

Ryan Elwell (M'10) received the B.S. degree in electrical and computer engineering from Rowan University, Glassboro, NJ, in 2008, and the M.S. degree in engineering from Rowan University in 2009.

He is currently a Technical Leader of radar applications in the U.S. Army Communications-Electronics Research, Development, and Engineering Center, Aberdeen, MD. His current research interests include neural networks, incremental learning, digital signal processing, and algorithm development for airborne radar exploitation.

Robi Polikar (SM'08) received the B.Sc. degree in electronics and communications engineering from Istanbul Technical University, Istanbul, Turkey, in 1993, and the M.Sc. and Ph.D. degrees both in electrical engineering and biomedical engineering, from Iowa State University, Ames, in 1995 and 2000, respectively.

He is currently a Professor of Electrical and Computer Engineering at Rowan University, Glassboro, NJ. His recent and current works are funded primarily through National Science Foundation's CAREER and Energy, Power and Adaptive Systems Programs. His current research interests include computational intelligence including ensemble systems, incremental and nonstationary learning, and various applications of pattern recognition in bioinformatics and biomedical engineering.

Dr. Polikar is a member of the American Society for Engineering Education, Tau Beta Pi, and Eta Kappa Nu.