Incremental Learning of Concept Drift from Streaming Imbalanced Data

Gregory Ditzler, *Student Member IEEE*, and Robi Polikar, *Senior Member IEEE*

Abstract—Learning in nonstationary environments, also known as learning concept drift, is concerned with learning from data whose statistical characteristics change over time. Concept drift is further complicated if the dataset is class-imbalanced. While these two issues have been independently addressed, their joint treatment has been mostly underexplored. We describe two ensemble-based approaches for learning concept drift from imbalanced data. Our first approach is a logical combination of our previously introduced Learn^{++}.NSE algorithm for concept drift, with the well-established SMOTE for learning from imbalanced data. Our second approach makes two major modifications to Learn^{++}.NSE-SMOTE integration by replacing SMOTE with a sub-ensemble that makes strategic use of minority class data; and replacing Learn^{++}.NSE and its class-independent error weighting mechanism with a penalty constraint that forces the algorithm to balance accuracy on all classes. The primary novelty of this approach is in determining the voting weights for combining ensemble members, based on each classifier's time and imbalance-adjusted accuracy on current and past environments. Favorable results in comparison to other approaches indicate that both approaches are able to address this challenging problem, each with its own specific areas of strength. We also release all experimental data as a resource and benchmark for future research.

Index Terms— incremental learning, concept drift, class imbalance, multiple classifier systems.

1 Introduction

Computational models of learning are typically developed for a particular problem domain, and optimized for specific conditions within that domain. These conditions usually dictate or restrict the amount and nature of the available data for training, the distributions from which such data are drawn, or the mechanism by which data become available, any of which can make it difficult to address multiple problems domains concurrently. The two problem domains featured in this paper, namely learning concept drift (i.e., learning in nonstationary environments) and learning from imbalanced data (i.e., with very few positive and many negative instances), are good examples, as there are well-established approaches for each. Many recent efforts – by us as well as other researchers – have separately focused on concept drift and class imbalance. A more general learning framework for accommodating the joint problem, that is, learning from a drifting (nonstationary) environment that also provides severely unbalanced data, is largely underexplored. With the omnipresence of realworld applications, such as climate monitoring, spam filtering, or fraud detection, the importance of developing a more general framework can hardly be overstated. For example, in spam identification problem, an official work related e-mail address may receive many legitimate and few spam e-mails. The goal is then to identify the minority class (spam) so that they can be removed. Conversely, a personal e-mail address may receive a large number of spams, but few work related e-mails, where the goal is then to identify the minority class (work related) e-mails so that they can be saved. Both cases are also concept drift problems, as the characteristics of both spam and legitimate e-mails change over time in part due to increasingly creative techniques used by spammers, and in part due to changing trends in user interest. Hence, this is an example of the joint problem of incremental learning of concept drift from class-imbalanced data.

Digital Object Indentifier 10.1109/TKDE.2012.136

Combining the definition of incremental learning, as suggested by several authors [1-3], along with Kuncheva's and Bifet's desiderata for nonstationary learning algorithms [4;5], we obtain the desired properties of a general framework for learning concept drift from imbalanced data as follows: (i) *Learning new knowledge*: building upon the current model using new data to learn novel knowledge in a wide spectrum of nonstationary environments; (ii) *Preserving previous knowledge*: determining what previous knowledge is still relevant (and hence should be preserved), what is no longer relevant (hence should be discarded / forgotten), but with the added ability to recall discarded information if the drift / change follow a cyclical nature; (iii) *One pass (incremental) learning*: learning one instance or one batch at a time without requiring access to previously seen data; and (iv) *Balance on minority /majority class performance*: maintaining high accuracy (*recall*) on minority class without sacrificing majority class performance.

This paper describes such a framework that includes two related ensemble-based incremental learning approaches, namely, Learn^{++}.CDS and Learn^{++}.NIE, neither of which place any restrictions on how slow, fast, abrupt, gradual, or cyclical the change in distributions may be. Both approaches are also designed to handle class imbalance, and are able to learn from new data that become available in batches, without requiring access to data seen in previous batches. The streaming and nonstationary nature of data strictly require incremental learning, which raises the so-called stability-plasticity dilemma, where "stability" describes retaining existing knowledge (for learning stationary subspaces or remembering recurring nonstationary distributions), and "plasticity" refers to learning new knowledge [6]. We show that an ensemble-of-classifiers based learning model that use carefully selected instances with strategically and dynamically assigned weights for combining member classifiers can indeed learn in such a nonstationary environment, and achieve a meaningful balance of stability and plasticity, even in the presence of class imbalance.

The primary contribution of this paper is such a general framework for learning from a stream of class-imbalanced data whose underlying distributions may be changing over time. This work complements our prior work on Learn^{++}.NSE (incremental learning for *N*on *S*tationary *E*nvironments) algorithm for learning concept drift. Learn^{++}.NSE trains a new classifier for each new batch of data, combining them using dynamically weighted majority voting, where voting weights are based on classifiers' time-adjusted errors averaged over recent environments [7]. However, Learn^{++}.NSE, like other algorithms not specifically designed to accommodate class-imbalance, becomes biased towards majority class in case of severe class imbalance. Two approaches are presented in this paper to develop a model that can learn concept drift from imbalanced data. The first is a natural combination Learn^{++}.NSE with the *S*ynthetic *M*inority class *O*versampling *TE*chnique (SMOTE), a well-established over-sampling approach that generates strategically positioned synthetic minority data. The second approach replaces Learn^{++}.NSE and its class-independent raw classification error with a new penalty con-

straint that simultaneously enforces both minority and majority class performance. It also replaces SMOTE with a bagging based sub-ensemble algorithm that makes strategic use of existing minority data.

In Sections II, III, and IV, we review the recent research on concept drift, class imbalance, and the joint problem, respectively, and describe the proposed approaches in detail in Section V. Section VI presents the experiments and results comparing the proposed approaches to existing ones on a variety of real world and carefully designed synthetic datasets. Concluding remarks, including a discussion on computational complexity of these approaches are provided in Section VII.

2 Concept Drift

In the context of machine learning, an environment that provides data whose joint distributions change over time, such that $p_{t+1}(\boldsymbol{x}, \omega) \neq p_t(\boldsymbol{x}, \omega)$, is referred to as a nonstationary environment (NSE). Here ω represents the class (concept) and $\boldsymbol{x}$ represents a data instance. Since class definitions change over time in a NSE, learning in such an environment is also referred to as *concept drift*. Much of early work in NSE learning have primarily focused on the definition of the problem, identifying the types of NSE [8-10] and the conditions under which they can be learned [11]. Characterizing such an environment is not trivial because the change can be abrupt or gradual, slow or fast, rare or often, random or systematic, cyclical or otherwise. Concept drift can also be perceived, rather than real, due to insufficient, unknown or unobservable features – referred to as *hidden context*, where an underlying phenomenon provides a true and static description over time for each class, which, unfortunately, is hidden from the learner. Having the benefit of knowing this (hidden) context would remove the nonstationarity. However, the learner can only make use of the information available, and in the absence of any information regarding its existence and nature, hidden contexts can only be modeled as a nonstationary environment.

Concept drift algorithms can be characterized in several ways, such as online vs. batch approaches depending on the number of instances used at each training step; single classifier vs. ensemble-based approaches depending on the number of classifiers used to make a decision; incremental vs. non-incremental approaches based on whether prior data are reused; or active vs. passive approaches depending on whether an active drift detection mechanism is employed. In active drift detection, the algorithm explicitly seeks to determine whether and when a change/drift has occurred before taking any corrective action. A passive drift detection algorithm, however, assumes that drift may occur at any time, or is continuously occurring, and hence updates a model every time new data arrive. Active approaches have the advantage of avoiding unnecessary updates when no drift is detected, but are prone to both type I and II errors, common on noisy data: the algorithm may fail to update a model when necessary or incorrectly update it when not necessary. Passive approaches avoid the problems associated with incorrect drift detection, but have increased computational burden due to constant update.

Several concept drift algorithms use some form of a sliding window over the incoming data, where the batch of in-

stances that fall within the window are considered stationary, and a new classifier is generated for each such batch of data. STAGGER [9] and FLORA [10] are the first examples of this passive batch-based *instance selection* approach. Certain versions of FLORA algorithms also include an active drift detection mechanism, using an adaptive window narrowing or widening depending on whether the drift is rapid or slow [10]. FLORA labels the classifiers as relevant, irrelevant or potentially relevant by evaluating them on the most recent data. Each classifier maintains a counter based on the number of correctly classified examples, and classifiers are pruned based on their relevance in the current data window. However, such an approach causes catastrophic forgetting [12]. Another group of active concept drift approaches are based on control charts, such as CUSUM (cumulative sum). Alippi and Roveri's *just-in-time* (JIT) classifiers [13-15], and their more recent *intersection of confidence intervals* (ICI) rule [16] are examples of such approaches. Information theoretic measures, such as entropy, mutual information, or Hoeffding bounds of individual features have also been used for detecting drift and updating a classifier, typically a decision tree [17-19]. Many of these approaches also include a FLORA-like windowing mechanism, including Hulten et al.'s concept adapting very fast decision tree (CVFDT) [20] or Cohen et al.'s incremental online-information network (IOLIN) algorithms [21;22].

Most drift detection approaches are generally quite successful in detecting abrupt changes, but may struggle with gradual drift. As an online active approach, the Early Drift Detection Method (EDDM) is specifically designed for gradual drift [23]. It does so by monitoring the distance between the errors of a classifier and comparing their mean to a threshold. EDDM has the ability to flag not only drift, but also issue a warning if necessary.

Multiple classifier systems (MCS), or *ensemble systems*, have also been proposed and successfully implemented for learning in nonstationary environments [24]. MCS provide a natural mechanism to update a knowledge base by adding, removing or updating classifiers. Most ensemble-based algorithms track the environment by adding new (and possibly removing old) classifiers to build an ensemble with each incoming dataset. These approaches typically use a passive drift detection along with a fixed ensemble size, where the oldest member (as in Street's Streaming Ensemble Algorithm (SEA) [25], and Bifet's adaptive Hoeffding tree bagging [26]) or the least contributing ensemble member (as in Tsymbal's Dynamic Integration [27], Kolter and Maloof's, Dynamic Weighted Majority (DWM) [28]) is replaced with a new one. Voting is the most common approach for combining the classifiers, though there is disagreement on what type of voting is most suitable for concept drift approaches. Tsymbal combines a proximity measure with classifier performance to determine the voting weights, with classifiers whose training and test data are in the same region of feature space being awarded higher weights [27]. Gao, on the other hand, advocates simple majority voting [29], pointing out that weights based on classifier error on drifting data is uninformative for future datasets. Other variations of ensemble approaches include [30-32].

Hybrid approaches that combine active detection, sliding windows and classifier ensembles have also been proposed, such as random forests with entropy [33] and Bifet's novel integration of a Kalman filter combined with an adaptive sliding window algorithm, namely `ADWIN` [5;34]. Bifet et. al. has recently released a WEKA-like software suite, Massive Online Analysis (MOA) at [35], which includes implementations of `ADWIN` and a variety of other tools for mining data streams with concept drift. Another example of active ensemble approach includes Hoens et. al.'s recent work that combines random subspace approach with Hellinger distance to detect drift in features of the data [36].

Ensemble based concept drift algorithms also include our previous work, the incremental Learn^{++}.NSE algorithm [7;37-40], which generates a new classifier with each batch of data that become available. The classifiers are combined via dynamically weighted majority voting, where weights are based on the time-adjusted error of each classifier, sigmoidally averaged over all environments. Learn^{++}.NSE can track a variety of environments, including those with gradual, fast, abrupt, or even cyclical drift. It can identify most and least relevant classifiers, assigning them high and low voting weights, and detect when a classifier becomes relevant again, should the environment follows a cyclic path. The algorithm can also accommodate addition or removal of classes, a property it shares with other members of Learn^{++} algorithms [3;12;41]. However, just like algorithms mentioned above, Learn^{++}.NSE is not equipped to address imbalanced data. This is because the class-independent raw classification error is a major contributor to the classifiers' weight, and with severely unbalanced datasets, classification error is a very poor estimator of the classifier's overall performance on minority data, which is typically the more critical figure of merit. Therefore, a new method of weighting classifiers and measuring their performance on all classes (minority and majority) is required to simultaneously address concept drift and class imbalance.

3 Class Imbalance in Machine Learning

Class imbalance occurs when a dataset does not have (approximately) equal number of examples from each class, which may be quite severe in some applications [42]. A comprehensive literature review of learning from imbalanced data can be found in [43]. The most obvious solution to class imbalance is under (over) sampling the majority (minority) class data. However, each has its own drawbacks: under-sampling throws away data from the majority class, whether they are useful or not. Over-sampling, on the other hand, creates exact replicates of the minority instances, which may cause the classifier to overfit the minority class instances. More intelligent approaches include *condensed nearest neighbor* (CNN) rule, which removes examples from the majority class that are distant from the decision border [44], or *Tomek links* that under-samples the majority class by defining a distance between two instances from different classes [45].

A more novel method, however, is used by Chawla's SMOTE (*Synthetic Minority Oversampling TEchnique*) algorithm [46], which populates the minority class feature space by strategically placing synthetic examples on the line segment con-

necting two minority instances (see Fig. 2). SMOTE has been shown to improve the classification accuracy on the minority class over other standard approaches. Alternatively, SMOTEBoost [47] combines SMOTE and AdaBoost.M2 to further improve *F*-measure and recall. More recently, bagging ensemble variation (BEV) was proposed [48], which uses bagging to train classifiers with all minority class data and subsets of the majority class data. Learn^{++}.UDNC combines preliminary confidence measures with a transfer function to adjusts the voting weights of classifiers based on the class imbalance in each dataset [3;12;49]. DataBoost-IM algorithm determines hard to classify minority class instances, based on which creates new synthetic data [50]. Finally, classifier specific approaches, such as rebalancing designed to work with support vector machines (SVMs), have also been recently proposed [51;52].

4 Previous Work on Combining Class Imbalance and Concept Drift

Recently, an algorithm for NSE learning from imbalanced data has been proposed by Gao et al. [32;53]. The algorithm, referred to as *uncorrelated bagging* (UCB), is based on a bagging framework that trains classifiers on a subset of the majority class instances (whose selection is controlled by a user defined parameter) and the union of all minority class instances seen thus far (current + previous positive examples). With each new batch of data, minority class instances are saved and accumulated for training the classifiers at the future iterations. However, this approach implicitly assumes that the minority class data are stationary. This assumption, when violated, can cause the algorithm to misinterpret the true feature space of the minority class at subsequent time steps. The issue of handling the accumulated data for lifelong learning is also not addressed. It is conceivable that the minority population could become majority as data are accumulated over long periods of time, and the data that have been accumulated could be irrelevant in the future. Furthermore, the approach cannot be formally considered incremental since it does not meet the single-pass requirement. Chen and He's *Selectively Recursive Approach* (SERA) algorithm uses a similarity measure to select previous minority examples that are most similar to those in the most recent dataset [54]. SERA is less prone to issues of the minority class drifting compared to UCB [53], and can be implemented in two ways: i) generate a single classifier on each dataset; or ii) use biased bagging, *BBagging,* that manually increases the sampling weights of the minority data. SERA effectively discards what it considers as irrelevant accumulated instances from the current training set by employing a (Mahalanobis) distance metric. SERA is modified to employ an ensemble approach in [55;56]. However, this framework is also not strictly incremental, as it requires access to previous data. Hence, both approaches listed above work best when the minority data concept is stationary and/or the old (minority class) data can be retained for future use. Finally, most recently, Xioufis et. al. have presented a window-based method that uses a *k*-NN for multi-label classification for data that contains concept drift and class imbalance [57].

Input: Training data $\mathfrak{D}^{(t)} = \{\boldsymbol{x}_i \in X; y_i \in \Omega\}$ where $i = 1,2,\dots,m^{(t)}$
Supervised learning algorithm **BaseClassifier**
Sigmoid parameters: a & b

for $t = 1,2,\dots$ **do**

1. Compute error of the existing ensemble
$$E^{(t)} = \sum_{i=1}^{m^{(t)}} \frac{1}{m^{(t)}} \cdot [\![H^{(t-1)}(\boldsymbol{x}_i) \neq y_i]\!] \quad (1)$$
2. Update and normalize instance weights
$$w^{(t)}(i) = \frac{1}{m^{(t)}} \cdot \begin{cases} E^{(t)} & H^{(t-1)}(\boldsymbol{x}_i) = y_i \\ 1 & otherwise \end{cases} \quad (2)$$
$$D^{(t)}(i) = w^{(t)}(i) \Big/ \sum_{j=1}^{m^{(t)}} w^{(t)}(j) \quad (3)$$
3. Call SMOTE on $\mathfrak{D}^{(t)}$ minority instances to obtian $\mathcal{S}^{(t)}$
4. Call **BaseClassifier** with $\mathfrak{D}^{(t)}$ and $\mathcal{S}^{(t)}$ to obtain: $h_t: X \to Y$
5. Evaluate existing classifiers on $\mathfrak{D}^{(t)}$ and obtain pseudo error
$$\varepsilon_k^{(t)} = \sum_{i=1}^{m^{(t)}} D^{(t)}(i) \cdot [\![h_k(\boldsymbol{x}_i) \neq y_i]\!] \quad (4)$$
if $\varepsilon_{k=t}^{(t)} > 1/2$, Generate new h_k, **end if**
if $\varepsilon_{k<t}^{(t)} > 1/2$, Set $\varepsilon_{k<t}^{(t)} = 1/2$ **end if**
$$\beta_k^{(t)} = \varepsilon_k^{(t)} / \left(1 - \varepsilon_k^{(t)}\right) \quad (5)$$
6. Compute weighted sum of normalized error
$$\sigma_k^{(t)} = 1/(1 + \exp(-a(t - k - b))) \quad (6)$$
$$\hat{\sigma}_k^{(t)} = \sigma_k^{(t)} \Big/ \sum_{j=0}^{t-k} \sigma_k^{(t-j)} \quad (7)$$
$$\hat{\beta}_k^{(t)} = \sum_{j=0}^{t-k} \hat{\sigma}_k^{(t-j)} \beta_k^{(t-j)},\ k = 1,2,\dots,t \quad (8)$$
7. Calculate voting weight
$$W_k^{(t)} = \log 1/\hat{\beta}_k^{(t)} \quad (9)$$
8. Compute ensemble decision
$$H^{(t)}(\boldsymbol{x}) = \arg\max_{c \in \Omega} \sum_{k=1}^{t} W_k^{(t)} [\![h_k(\boldsymbol{x}_i) = c]\!] \quad (10)$$

end for

Output: Return final hypothesis $H^{(t)}(\boldsymbol{x})$

Figure 1. Learn^{++}.CDS pseudo code

Input: Minority data $\mathfrak{D}^{(t)} = \{\boldsymbol{x}_i \in X\}$ where $i = 1,2,\dots,T$
Number of minority instances (T), SMOTE percentage (N), number of nearest neighbors (k)

for $i = 1,2,\dots,T$ **do**
1. Find the k nearest (minority class) neighbors of $\boldsymbol{x}_i$
2. $\widehat{N} = \lfloor N/100 \rfloor$

while $\widehat{N} \neq 0$ **do**
1. Select one of the k nearest neighbors, call this $\overline{\boldsymbol{x}}$
2. Select a random number $\alpha \in [0,1]$
3. $\widehat{\boldsymbol{x}} = \boldsymbol{x}_i + \alpha(\overline{\boldsymbol{x}} - \boldsymbol{x}_i)$
4. Append $\widehat{\boldsymbol{x}}$ to $\mathcal{S}$
5. $\widehat{N} = \widehat{N}$ 1

end while

end for

Output: Return synthetic data $\mathcal{S}$

Figure 2. SMOTE pseudo code

5 Proposed Framework for Learning Concept Drift from Imbalanced Data

A general framework for learning concept drift from streaming imbalanced data should satisfy the aforementioned desired properties: accommodate a variety of concept drift environments (e.g., slow or rapid, gradual or abrupt, cyclical or otherwise); learn from possibly severely imbalanced data with good performance on both minority and majority class data; retain and extract knowledge from past experience when such knowledge is relevant; and learn incrementally without access to the previous data. In this section, we describe two algorithms that meet these criteria.

5.1 Learn^{++}.CDS

We start with two approaches, each addressing the individual problem for which they were originally designed: Learn^{++}.NSE for learning concept drift [7] and SMOTE for class imbalance [46]. We choose Learn^{++}.NSE primarily due to its versatility: its ability to accommodate a variety of concept drift scenarios, including cyclic or variable rate of drift, as well as class addition / removal that most other algorithms do not address. For imbalanced data, we choose SMOTE whose ability to learn from severely imbalanced datasets has been well documented. Therefore, a natural first step would

be to suitably combine the strengths of these two algorithms. In such a setting where imbalanced data are received from a nonstationary environment, SMOTE can first be used to reduce the imbalance ratio and enrich the minority class feature space, followed by Learn^{++}.NSE to learn the drifting concept from the newly *rebalanced* dataset [58]. We call this approach Learn^{++}.CDS (Learn^{++} for *Concept Drift with SMOTE*), whose pseudo code is shown in Fig. 1.

Learn^{++}.CDS is provided with dataset $\mathfrak{D}^{(t)}$ at time step t, drawn from the distribution $p_t(\boldsymbol{x}, \omega)$ that may be different from $p_{t-1}(\boldsymbol{x}, \omega)$. The algorithm then updates a distribution of instance weights, $D^{(t)}$, by evaluating the existing ensemble, $H^{(t-1)}(\boldsymbol{x}_i)$, on the most recent dataset, $\mathfrak{D}^{(t)}$ (Step 1). The instance index is given by $i = 1, \ldots, m^{(t)}$ where $m^{(t)}$ is the number of labeled instances at time step t. The weights of the misclassified instances are increased and renormalized to create a penalty distribution $D^{(t)}$ (Eq. (2), (3) in Step 2) to be later used in computing classifiers' pseudo errors. A new classifier, hypothesis h_t, is trained on $\mathfrak{D}^{(t)}$ that is enriched by synthetic minority class samples $\mathcal{S}^{(t)}$ provided by SMOTE (Steps 3 & 4). The pseudo code for SMOTE, adopted from [46], can be found in Fig. 2.

All classifiers generated up to time t are evaluated on $\mathfrak{D}^{(t)}$ to obtain their pseudo errors on the new environment (Eq. (4), Step 5) through a weighted sum of the penalty distribution. Three critical points are worth mentioning here. First, since classifiers are generated at different times, each classifier receives a different number of evaluations: at time t, h_t is evaluated for the first time, whereas h_1 gets its t^{th} evaluation. We use $\varepsilon_k^{(t)}, k = 1, \ldots, t$ to denote the pseudo error of h_k, the classifier generated at time step k, on dataset $\mathfrak{D}^{(t)}$. Henceforth, when applicable, the superscript represents the time index for the current environment, and the subscript is the time the relevant classifier is generated.

Second, not all misclassifications contribute equally to the pseudo error $\varepsilon_k^{(t)}$, as the penalty distribution $D^{(t)}$ is used to weigh the misclassifications. In this weighting scheme, the relativity of penalties is based on the overall ensemble error. When the ensemble performs well on the new data – indicating that there has been little or no change in the underlying distributions – misclassified instances 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 smaller 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. Note that the weights are not used for instance selection, as done in many other boosting approaches [59]. In fact, unlike most concept drift algorithms, there is no instance selection in Learn^{++}.CDS, as all data in the current batch are used for training. Rather, the instance weights are used to control penalty assignment, which is then used to determine voting weights. The goal in this formulation is to allow the ensemble learn new knowledge in $\mathfrak{D}^{(t)}$, while reinforcing still relevant existing knowledge.

Third, if the pseudo error for the newest classifier on its own training data satisfies $\varepsilon_{k=t}^{(t)} \geq ½$, it is deemed an ineffec-

tive classifier and is replaced with a new classifier. However, an earlier classifier with $\varepsilon_{k<t}^{(t)} \geq ½$, is simply has its error set to ½, which - when normalized (later in step 7) - receives the highest deemed error (and hence the lowest vote) at time stamp t. Note that while the new classifier with an error $\varepsilon_{k=t}^{(t)} \geq ½$ is discarded, an earlier classifier with a similar error is retained (but assigned a zero weight for that environment). This is because an older classifier currently underperforming may become relevant again if the earlier environment on which it performed well recurs. These errors $0 \leq \varepsilon_k^{(t)} \leq ½$ are then normalized to obtain $0 \leq \beta_k^{(t)} \leq 1$ (Eq. (5) in Step 5). The normalized errors are further weighted to emphasize classifiers' recent performance, using a sigmoidal weighting function (Step 6), which also serves to reduce the effects of wide swings in errors, possibly due to outliers or inherent noise in the data. The sigmoid parameter a defines the slope of the sigmoid cutoff, and b refers to the number of prior pseudo errors to be considered before the cutoff. Such a time adjustment approach rewards classifiers that are currently performing well on the most recent environments, even if such classifiers may have been generated long time ago. Classifiers with high pseudo error on current environments then receive (near) zero weight, temporarily removing them from affecting the decision. We note that if the same classifier later performs well on a future environment, it is automatically reactivated by receiving a higher voting weight.

Final voting weights are computed as the logarithm of the reciprocals of the time-adjusted weighted classifier errors in Step 6 (Eq. (9), Step 7). The final ensemble decision is then obtained using weighted majority voting (Eq. (10), Step 8).

5.2 Learn^{++}.NIE

The error metric used for the penalty distribution in Learn^{++}.CDS does not discriminate between instances of different classes. In an imbalanced dataset, such a class-independent error measurement causes the error to be biased towards the majority class. Furthermore, in an imbalanced dataset, simple raw classification accuracy (RCA) is misleading, as it can lead to low overall error by correctly classifying majority class instances and missing all minority class instances. Yet, it is the performance on the minority class data (also called *recall*) that is usually of utmost importance. Learn^{++}.CDS addresses this issue by pre-balancing the originally imbalanced data with synthetic samples generated by SMOTE. An alternate approach, however, is to optimize the classifier performances on both minority and majority class data, without generating synthetic data. Learn^{++}.NIE (Learn^{++} for *Nonstationary and Imbalanced Environments*) follows such an approach [60].

Learn^{++}.NIE makes two strategic modifications compared to Learn^{++}.CDS. First, it employs a different penalty constraint that forces the algorithm to balance predictive accuracy on all classes by rewarding classifiers that perform well on *both* minority and majority class data. Furthermore, this constraint can be adjusted to place more or less weight to minority or majority class instances, depending on the application, balancing minority recall while preserving a majority

Input: Training data $\mathfrak{D}^{(t)} = \{\boldsymbol{x}_i \in X; y_i \in \Omega\}$ where $i = 1,2,\dots,m^{(t)}$
Supervised learning algorithm **BaseClassifier**
Sigmoid parameters: a & b
Error weight: $0 \le \eta \le 1$
Ensemble size: T

for $t = 1,2,\dots$ **do**

1. Call $H_t = BaggingVariation(\textbf{BaseClassifier}, \mathfrak{D}^{(t)}, T)$
2. Evaluate all existing sub-ensembles on $\mathfrak{D}^{(t)}$ to determine classifier weight measure $\varepsilon_k^{(t)}$ using (17), (18), or (19).

   if $\varepsilon_{k=t}^{(t)} > 1/2$, Generate new sub-ensemble; **end**
   if $\varepsilon_{k<t}^{(t)} > 1/2$, Set $\varepsilon_{k<t}^{(t)} = 1/2$ **end if**

$$\beta_k^{(t)} = \varepsilon_k^{(t)} / \left(1 - \varepsilon_k^{(t)}\right) \quad (11)$$

3. Compute weighted sum of normalized error where $k = 1,2,\dots,t$

$$\sigma_k^{(t)} = 1/(1 + \exp(-a(t-k-b))) \quad (12)$$

$$\hat{\sigma}_k^{(t)} = \sigma_k^{(t)} \Big/ \sum_{j=0}^{t-k} \sigma_k^{(t-j)} \quad (13)$$

$$\hat{\beta}_k^{(t)} = \sum_{j=0}^{t-k} \hat{\sigma}_k^{(t-j)} \beta_k^{(t-j)} \quad (14)$$

4. Calculate voting weight

$$W_k^{(t)} = \log \frac{1}{\hat{\beta}_k^{(t)}} \quad (15)$$

5. Compute ensemble decision

$$H^{(t)}(\boldsymbol{x}) = \arg\max_{c \in \Omega} \sum_{k=1}^{t} W_k^{(t)} [\![H_k(\boldsymbol{x}_i) = c]\!] \quad (16)$$

end for

Output: Return final hypothesis $H^{(t)}(\boldsymbol{x})$

Figure 3. Learn++.NIE pseudo code

Input: Training data $\mathfrak{D} = \{\boldsymbol{x}_i \in X; y_i \in \Omega\}$ where $i = 1,2,\dots,m$
Supervised learning algorithm **BaseClassifier**
T: number of classifiers to generate

Split $\mathfrak{D}$ into $\mathfrak{D}_+$ and $\mathfrak{D}_-$ corresponding to the minority and majority data
for $k = 1,2,\dots,T$ **do**

1. Randomly draw $\lfloor N/T \rfloor$ instances from $\mathfrak{D}_-$. Call this $\widehat{\mathfrak{D}}$
2. Call BaseClassifier with $\{\mathfrak{D}_+, \widehat{\mathfrak{D}}\}$ to form a hypothesis $h_k: X \to \Omega$

end for
Compute composite hyopthesis for an unlabeled instance $\boldsymbol{x}$

$$H(\boldsymbol{x}) = \arg\max_{c \in \Omega} \sum_{k=1}^{T} \frac{1}{T} [\![h_k(\boldsymbol{x}) = c]\!]$$

Output: Return composite hypothesis $H(\boldsymbol{x})$

Figure 4. BaggingVariation pseudo code

class performance. Second, Learn^{++}.NIE also replaces the minority class oversampling of SMOTE with that of a bagging based sub-ensemble, which neither oversamples existing minority data, nor generates synthetic data, as described below.

The pseudo code for Learn^{++}.NIE is shown in Fig. 3. The algorithm employs a variation of bagging (*BaggingVariation*) to generate sub-ensembles of classifiers (Step 1). *BaggingVariation*, shown in Fig. 4, first divides the training data, $\mathfrak{D}^{(t)}$, into the majority and minority classes. A sub-ensemble of classifiers is generated, each using all of the minority class data and a randomly sampled subset of the majority class. A few points are worth noting here. First, no minority data instance is ever repeated for balancing class proportions in training any of the classifiers, and unlike SMOTE, no synthetic minority class data are generated either. The strategy here is to have each classifier being exposed to a balanced dataset that uses all the minority class information and a subset of majority class information. Second, unlike the standard under sampling approaches, majority class data are not discarded, as each ensemble member is trained on a different bootstrap sample of the majority class data. Third, minority (or, for that matter, majority) class data are not accumulated across time: each minority class instance is only used on the sub-ensemble generated at the current time t; therefore the implicit – and possibly incorrect – assumption of minority class data being stationary is *not* made by Learn^{++}.NIE. Finally, since data instances are only used for generating classifiers at the current time they become available, and are never reused in future classifiers,

Learn^{++}.NIE meets the strict incremental learning one-pass criterion. The classifiers in each sub-ensemble are then combined via simple majority voting to form a composite hypothesis. The composite hypothesis obtained from the sub-ensemble at time t is denoted by $H^{(t)}$ in the pseudo code in Fig. 3.

All existing sub-ensembles (H_k for $k = 1,2,\dots,t$) are evaluated on $\mathfrak{D}^{(t)}$. Learn^{++}.NIE then computes the error measure $\varepsilon_k^{(t)}$ to be used in determining the weight of the sub-ensemble. As mentioned above, this measure is not based on the raw classification accuracy, but designed to accommodate the imbalanced nature of the data. Specifically, Learn^{++}.NIE uses one of three carefully selected measures, the first of which is the *weighted recall measure* (*wrm*) for boosting recall and tracking drifting concepts. The *wrm* measure is determined by computing the weighted average of recall on the majority class ($p_{k,(-)}^{(t)}$) and minority class ($p_{k,(+)}^{(t)}$). The weighted error $\varepsilon_k^{(t)}$ is computed using (17) from the performances of the majority and minority classes, hence *wrm* is a convex combination of the recall of the majority and minority classes. The term η controls the weight given to a particular class error and is bound between zero and one ($\eta \in [0,1]$). Therefore, we can control the penalty incurred for the error of a class rather than penalizing for the misclassification of a particular instance.

$$\varepsilon_k^{(t)} = \eta\left(1 - p_{k,(+)}^{(t)}\right) + (1-\eta)\left(1 - p_{k,(-)}^{(t)}\right) \tag{17}$$

Selecting $\eta = 1$ (or $\eta = 0$) penalizes a classifier for error on the minority (or majority) class only. Thus, we would expect the ensemble to have a very high minority (or majority) class performance, but poor overall accuracy because a sub-ensemble is not penalized for not learning the majority (or minority) class. Selecting $\eta = 0.5$ typically provides a good balance between minority class performance and the overall accuracy.

Two additional figures of merit that can be used to accommodate imbalanced data via Learn^{++}.NIE are the *geometric mean* (*G-mean*) and *F-measure*, which are commonly used in assessing classifier performance on imbalanced data. In using the former, we acknowledge that sub-ensembles that perform well across all classes should have a larger geometric mean than classifiers that perform poorly on any given class. For example, a sub-ensemble may have a relatively high classification accuracy but a poor *G*-mean, if it performs poorly on a minority class. In other words, G-mean indicates if a classifier is performing well across all classes or just a majority class. When the *G*-mean measure is used, $\varepsilon_k^{(t)}$is computed as follows:

$$\varepsilon_k^{(t)} = 1 - \prod_{j=1}^{C}\left[p_{k,j}^{(t)}\right]^{\frac{1}{C}} \tag{18}$$

where $p_{k,j}^{(t)}$ indicates the performance of sub-ensemble $H^{(k)}$ on class j instances at time t.

The third metric, the *F*-measure, explicitly tries to balance precision and recall performance, as shown in Eq. (19),

$$\varepsilon_k^{(t)} = 1 - F_1 \tag{19}$$

$$F_1 = 2\left(\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}\right) \tag{20}$$

where F_1 is the F_1-score or F-measure given by Eq.(20). The strategy behind using measures like the *F*-measure, *G*-mean and weighted recall measure is to allow the algorithm to weigh classifiers on how well they are performing across *all* classes, majority and minority. Thus, if a sub-ensemble is consistently performing poorly on a minority (or majority) class, then the final voting weight, $W_k^{(t)}$ as computed in Eq. (15), will reflect this sub-par performance. Once the sub-ensemble errors $\varepsilon_k^{(t)}$ are determined, the rest of Learn^{++}.NIE follows as Learn^{++}.CDS, but with individual classifiers h_t of Learn^{++}.CDS being replaced with the corresponding sub-ensembles H_t. We note that while individual classifiers in a sub-ensemble are combined via simple majority voting, the sub-ensemble decisions themselves are combined via a weighted majority vote (Eq. 16) to obtain the final hypothesis, where voting weights are ultimately based on one of three metrics described above.

6 Experiments

In this section we present an empirical analysis of the Learn^{++}.NIE and Learn^{++}.CDS, including comparisons to other approaches recently proposed for nonstationary learning from imbalanced data. The experiments include a number of carefully designed synthetic datasets, as well as real world problems. The synthetic datasets are particularly useful as we can custom build a variety of concept drift scenarios to determine whether the proposed approaches can successfully address the joint concept drift – imbalanced data problem under such scenarios. The comparisons also allow us to determine the individual strength and weakness of each algorithm, compared to other existing approaches on a broad spectrum of concept drift scenarios. There are four synthetic datasets (drifting Gaussians, rotating checkerboard, shifting hyperplane and rotating spirals) and two real world datasets: commonly used electricity dataset, and a new weather prediction dataset. We release all data at [61] as a resource to all researchers interested in this field. The existing approaches against which Learn^{++}.CDS and Learn^{++}.NIE are compared include SEA, SERA, uncorrelated bagging and Learn^{++}.NSE.

6.1 Datasets and Preprocessing

All experiments begin at time t = 0 and end at some arbitrary time in the future, during which T consecutive batches of data are presented for training. Each batch is drawn from a possibly different source distribution, and the nature of drift between any two arbitrary time stamps is assumed unknown. The nature and rate of *change in drift* are also assumed unknown to the algorithm. Results of Learn^{++}.NSE using various effective drift rates (i.e., T values) can be seen in our prior work [38]. In general, as expected, the ability of the algorithm to track the drift is inversely proportional to the rate of drift.

The *drifting Gaussians dataset* has a majority class, comprised of a linear combination of three Gaussian components and a minority class drawn from a single Gaussian component. Table 1 presents the parametric equations that govern the movement of the mixtures over the time interval, $0 \le t \le 1$, where $C_{i,j}$ represents j^{th} Gaussian component for class ω_i.

Class 1 and class 2 are majority and minority classes, respectively. The off-diagonal elements of the covariance matrix for all classes are zero. The drift is controlled by varying the mean and covariance of each Gaussian over the duration of the experiment. The drift rate was set to 0.01 (T=100 time stamps), resulting in moderate drift rate. Since this is a controlled experiment of known distribution, we can also compare all algorithms to the optimal Bayes classifier trained only on the current data. The posterior probability of the optimal Bayes classifier at four different time stamps is shown in Fig. 5. Note that only the covariance matrices drift during $0 \leq t < t_1$, while the mean vectors remain constant. Then, the component means begin to drift and the location of the minority class eventually drifts into the center of the three-majority class mixtures as shown in the posterior estimate of Fig. 5(a). Finally, the minority class mode moves out from the center of the three majority class modes. The cardinality of the minority class (both training and testing) was set to ≈ 3% of total data.

The *rotating checkerboard problem* is a challenging generalization of the classical non-linear XOR problem. The XOR problem is the special case when for $\alpha \in \{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi\}$. The experiment is controlled by two parameters: the relative side length of each square with respect to the total length of the board (fixed at 0.5 in our experiments) and the angle of rotation, α. By varying the rotation angle in smaller steps between 0 and 2π, a significantly more challenging set of decision boundaries can be created. This experiment is particularly effective for observing an algorithm's ability to learn data in recurring environments. Fig. 6 depicts six snapshots of the board, corresponding to half of the rotating ($\alpha \in [0, \pi]$) checkerboard experiment. During the second half ($\alpha \in [\pi, 2\pi]$), the environment identically repeats the first half, which in turn simulates a recurring environment. The minority class consists of ~5% of the total data size.

The *shifting hyper-plane* problem, now a benchmark in concept drift problems, was introduced by Street & Kim for their *streaming ensemble algorithm* (SEA) [25]. In our implementation, the shifting hyper-plane in [25] was slightly modified to induce an imbalanced class distribution as well as a recurring environment. 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, only two of which are 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 the hyperplanes: an instance is assigned to class 1 if the sum of its relevant features ($x_1 + x_2$) fall below the threshold θ, 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 is changed three times throughout the experiment (4→7→4→7). We then induced class imbalance that also varies as the hyperplane shifts, to include minority data cardinality between ≈ 7% and ≈ 25% of total data size. Data also contain 5% class label noise. Datasets are presented in batches of 1000 instances for a total of T = 200 time stamps, with three different shifts occurring every 50 time stamps.

Table 1. Drifting Gaussians dataset parametric equations for the drift scenario

	$t = 0$ *to* $t = 1/3$				$t = 1/3$ *to* $t = 2/3$				$t = 2/3$ *to* $t = 1$			
	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y
$C_{1,1}$	1	$1+6t$	2	5	1	3	2	5	1	$8-9\left(t-\frac{1}{3}\right)$	$8-9\left(t-\frac{1}{3}\right)$	$8-9\left(t-\frac{1}{3}\right)$
$C_{1,2}$	$3-6t$	1	5	8	1	1	$5+9\left(t-\frac{1}{3}\right)$	8	1	1	8	8
$C_{1,3}$	$3-6t$	1	5	2	1	1	$5+9\left(t-\frac{1}{3}\right)$	2	1	1	8	2
$C_{2,1}$	1	1	8	5	1	1	$8-9\left(t-\frac{1}{3}\right)$	5	1	1	$8-9\left(t-\frac{1}{3}\right)$	$8-9\left(t-\frac{1}{3}\right)$

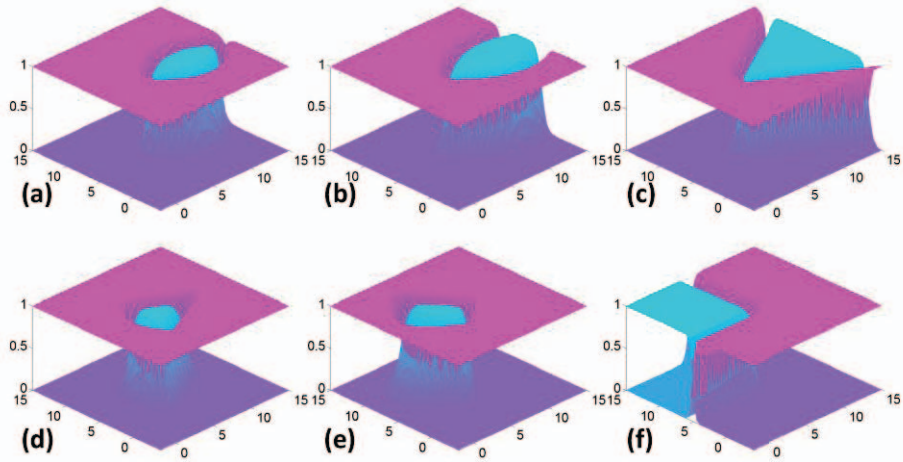

Figure 5. Posterior estimate of the Gaussian drift problem at six different time stamps. The cyan region represents the minority and pink represents the majority regions. Each portion of the figure represents the posterior at a point in time starting at $t = 1$ and moving to $t = [20,40,60,80,100]$.

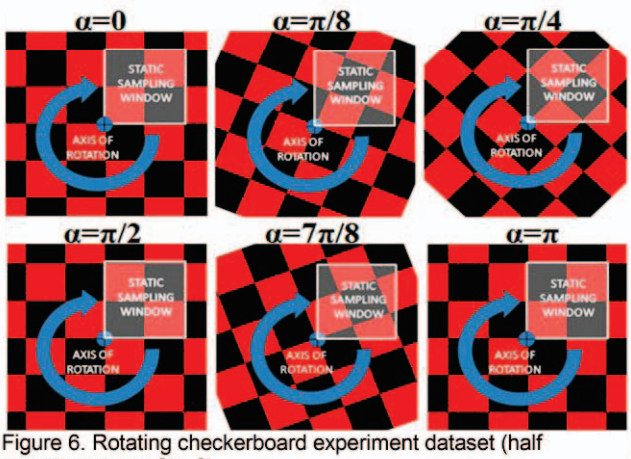

Figure 6. Rotating checkerboard experiment dataset (half experiment, $\alpha \in [0,\pi]$)

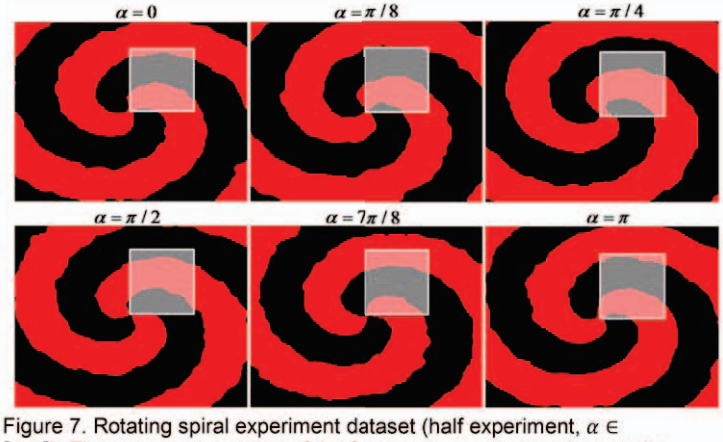

Figure 7. Rotating spiral experiment dataset (half experiment, $\alpha \in [0,\pi]$). The transparent region of the feature space represents a static window throughout the course of the experiment.

The *rotating spiral dataset* consists of four spirals, two for the minority and two for the majority class, as shown in Fig. 7. This figure shows the true decision boundaries at six different time stamps where the angle, α, varies between $[0,\pi]$. The initial feature space for the rotating spiral dataset is shown in Fig. 7(a) and the spirals rotate over 300 time stamps at evenly spaced intervals. The environment repeats every 150 time stamps (i.e. $\alpha = \pi$), thus the experiment contains two full rotations, again to simulate a recurring environment. The minority class consists of ~5% of the total data size.

The *Electricity Pricing dataset* (elec2) was first introduced in [62] and has also been used as a benchmark for concept drift problems [23;28]. The dataset provides time and demand fluctuations in the price of electricity in New South Wales, Australia. We use the day, period, NSW (New South Wales) electricity demand, VIC (Victoria) electricity demand and the

scheduled electricity transfer as the features. The task is to predict whether the NSW price will be higher or lower than VIC's in a 24 hour period. Instances with missing features were removed. Since the original dataset does not contain class imbalance, one of the classes was under sampled to create an imbalance ratio of approximately 1:18 (minority data is 5.5% of total data size). This dataset includes natural concept drift, as the electricity prices change with demand in time.

Finally, *the Weather Dataset* is a subset of the NOAA data [63] that we first processed and released as a concept drift dataset. While the dataset contains weather data from hundreds of locations around the world, we chose Offutt Air Force Base in Bellevue, Nebraska because this location contained over 50 years worth of data, providing not only cyclical seasonal changes, but also possibly long-term climate change. Daily measurements were taken for a variety of features such as temperature, pressure, visibility, and wind speed. We chose eight features and set the classification task to predict whether rain precipitation was observed on each day. We use a test-then-train strategy for evaluating the algorithms to cast this as a prediction problem, rather than a pure classification task. The training size was set to 120 instances (days), approximately one season, and data from the next season served as the test data. Minority data cardinality varied between 10% and 30% throughout the 50-year data.

6.2 Algorithms for Comparison and Parameter Selection

We use several state-of-the-art algorithms to compare their ability to learn concept drift from imbalanced data. The selection of algorithms include two that are designed strictly for concept drift (SEA and Learn^{++}.NSE) and others that are designed to handle concept drift and class imbalance. SEA and Learn^{++}.NSE are included to serve as baseline comparisons to demonstrate why new algorithms are needed to handle concept drift and class imbalance, and to demonstrate possible repercussions of blindly applying concept drift algorithms to imbalanced data. Learn^{++}.NSE comparison allows us to determine the benefit in Learn^{++}.CDS when SMOTE is applied for rebalancing the data in a nonstationary environment. The proposed Learn^{++}.CDS and Learn^{++}.NIE algorithms are compared to the following approaches:

- *Streaming Ensemble Algorithm (SEA)*: Benchmark concept drift algorithm. See [25] for implementation details. Ensemble size was limited to 25 (same as suggested by the paper's authors) for all results presented here.
- *Learn^{++}.NSE*: Predecessor of our proposed approach, originally designed for concept drift problems. See [7] for implementation details. The a and b parameters were set to 0.5 and 10, respectively.
- *Selectively Recursive Approach (SERA)*: Recently proposed for imbalanced datasets in concept drift problems. See [54] for implementation details. Sample size parameter, f, was set to 0.4 and *BBagging* used five classifiers [54].
- *Uncorrelated Bagging (UCB)*: Recently proposed for imbalanced datasets in concept drift problems [53]. The majority class parameter, r, was set to 0.4 for all experiments, and the number of classifiers in the ensemble was set to 5.

We used classification and regression tree (CART) as the base classifier for all algorithms for a fair comparison. The

SMOTE parameter for Learn^{++}.CDS was set based on the imbalance in the dataset. The η parameter of Learn^{++}.NIE for the weighted error combination was set to 0.5 which gives equal weight to the error of each class. Further analysis of the impact of this parameter is discussed in [60]. The under-sampling in the bagging variation for Learn^{++}.NIE was naturally set to N/K where N is the number of instances in a batch and K is the number of classifiers generated at each time stamp. We generate K=5 classifiers for each sub-ensemble of Learn^{++}.NIE. For NOAA dataset, however, the under-sampling ratio was set to 65% of the batch size as this dataset does not contain a large class imbalance compared to the synthetic datasets. We use Learn^{++}.NIE (*wrm*) to refer to Learn^{++}.NIE using (17) for the error measure $\varepsilon_k^{(t)}$. Similarly, Learn^{++}.NIE (*fm*) and Learn^{++}.NIE (*gm*) use Eq. (18) and (19) to determine $\varepsilon_k^{(t)}$. The a and b parameters for Learn^{++}.NIE and Learn^{++}.CDS's sigmoid were set to 0.5 and 10, respectively. To avoid the accumulated minority data in UCB becoming a majority class (since this issue is not addressed in the original UCB algorithm), we apply a sliding window to the minority class data to ensure that we do not train on more positive (minority) instances than negative (majority) ones. Otherwise, original UCB performs even poorer than the results provided below.

6.3 Evaluation Measures

Several figures of merit are used for a thorough evaluation of the algorithms. First, while raw classification accuracy (RCA) - % of all instances correctly identified – is an important and commonly used metric, it is not an adequate evaluation measure with imbalanced datasets. Therefore, we also include the *F*-measure, area under the receiving operating characteristic curve (AUC), and minority class recall to measure the effectiveness of the competing algorithms. All metrics are obtained at each time step, creating a time-series data (presented below). We also average each figure of merit over the course of the entire experiment to form a single value that represents how well an algorithm performs on a particular dataset. The averages (i.e., *F*-measure, recall, AUC, RCA, and OPM) are then ranked to make comparisons among algorithms. The ranks can range from (1) to (8) where (1) is the best and (8) is the worst performing algorithm. We then use the Friedman test as described in [64] to make formal statistical comparisons between classifiers over multiple datasets after all datasets are presented. Since no single algorithm may outperform all others in all measures and all datasets, we also use a combined *overall performance measure* (OPM) and *average rank* as composite figures of merit. OPM is simply a convex combination of raw classification accuracy, *F*-measure, AUC and minority class recall:

$$\text{OPM} = \lambda_1 \times \text{RCA} + \lambda_2 \times F_1 + \lambda_3 \times \text{AUC} + \lambda_4 \times \text{recall} \tag{21}$$

with $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \frac{1}{4}$ for the purposes of this study. We selected recall, and not precision, for the calculation of OPM since the *F*-measure is already included in the calculation, and itself is the harmonic mean of precision and recall.

6.4 Experimental Results

The time-series representations of the aforementioned figures of merit are first presented below. The shading enveloping each curve in the figures below represents a 95% confidence interval based on 40 independent trials unless otherwise noted (*refer to the digital copy of the article for color figures*). Due to the number of comparisons (6 algorithms and 8 variations on several different figures of merit) for each database, we split the results into two sets of figures: i) comparing three versions of Learn^{++}.NIE with respect to three different error measures (F-measure (*fm*), G-mean (*gm*) and weighted recall measure – (*wrm*)); and ii) comparing Learn^{++}.CDS, Learn^{++}.NIE (*fm*), UCB and SERA. Among three versions of Learn^{++}.NIE, F-measure was used as the representative in comparison against other algorithms, since F-measure combines both recall and precision. The SEA and Learn^{++}.NSE performances have been omitted in the figures due to space limitations; however, their results are included in the tables below which show the time-averaged figures of merit, their 95% confidence intervals and rankings for all algorithms. Before we present the detailed analysis of the experimental results, we first summarize our key observations that were consistent across a wide range of datasets and learning scenarios.

- Learn^{++}.NIE (*fm*) and Learn^{++}.CDS consistently provide results that rank them in the top three for the composite figure of merit (i.e., the overall performance measure - OPM) on nearly all datasets tested.
- Learn^{++}.NIE (*fm*) and Learn^{++}.CDS typically provide a significant improvement in recall, AUC, and OPM compared to their predecessor Learn^{++}.NSE and SEA, while maintaining good raw classification accuracy. UCB usually shows the best recall, but that comes at the cost of classification accuracy and F-measure. In fact, while UCB consistently maintains a good rank for recall, it holds the worst rank in terms of overall accuracy. Learn^{++}.NIE (*fm*) and Learn^{++}.CDS provide significant improvement in classification accuracy and F-measure compared to UCB.
- The simple integration of SMOTE into Learn^{++}.NSE has improved the OPM rank on every dataset over Learn^{++}.NSE, indicating that a blind use of even a strong concept drift algorithm is ill-advised for imbalanced data.
- Learn^{++}.NIE (*fm*) typically provides better results than its (*gm*) or (*wrm*) implementations. We find that Learn^{++}.NIE (*fm*) provides significant improvements over (*wrm*) in AUC, recall and OPM, whereas only AUC is improved when compared to (*gm*). On the other hand, the *wrm* implementation provides a unique control over recall or precision performance, a capability that the other two implementations do not have.

6.4.1 Drifting Gaussians Dataset

The results for the drifting Gaussians dataset are presented in Fig. 8 and 9. The average values of the figures of merits used in the evaluation are tabulated in Table 2 along with the results from SEA and Learn^{++}.NSE. There are several observations we can make from these results. First, all algorithms experience a major drop in every measure around time step 50, which

precisely corresponds to the minority class distribution moving into the middle of the majority class components, making

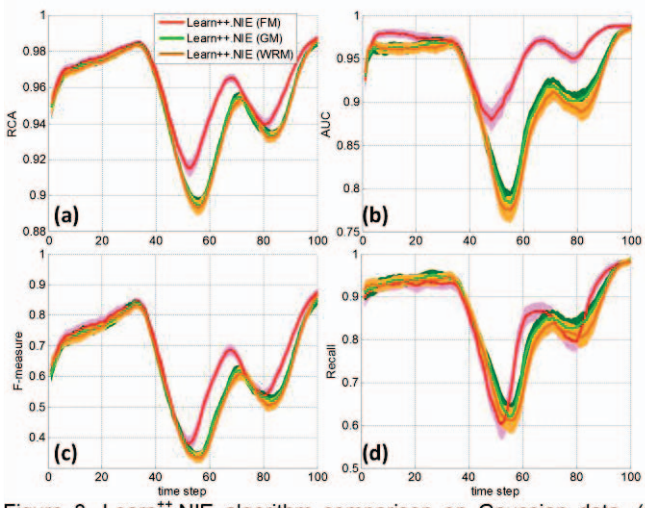

Figure 8. Learn^{++}.NIE algorithm comparison on Gaussian data. (a) RCA, (b) AUC, (c) F-measure, and (d) Recall

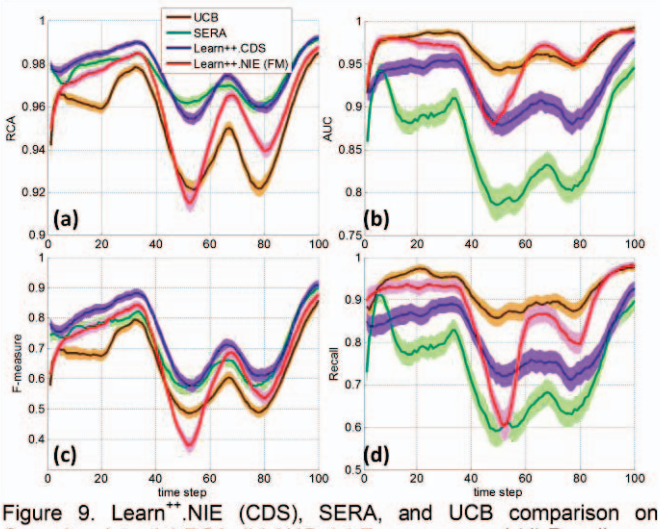

Figure 9. Learn^{++}.NIE (CDS), SERA, and UCB comparison on Gaussian data. (a) RCA, (b) AUC, (c) F-measure, and (d) Recall

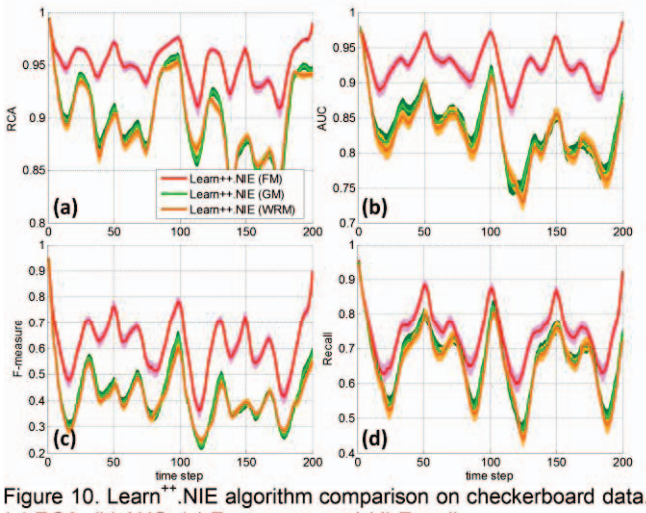

Figure 10. Learn^{++}.NIE algorithm comparison on checkerboard data. (a) RCA, (b) AUC, (c) F-measure, and (d) Recall

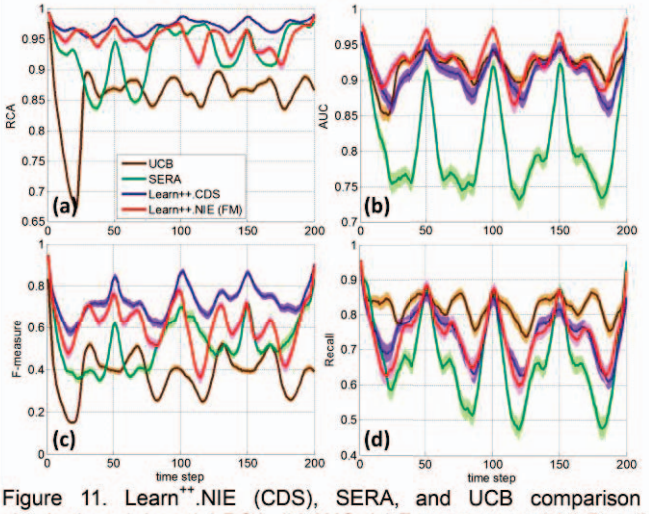

Figure 11. Learn^{++}.NIE (CDS), SERA, and UCB comparison on checkerboard data. (a) RCA, (b) AUC, (c) F-measure, and (d) Recall

prediction of minority class the most difficult. UCB appears to be the most robust in terms of recall and AUC, with Learn^{++}.NIE catching up in the latter part of the simulation. However, Learn^{++}.NIE (*fm*) maintains a better average rank (closely followed by Learn^{++}.CDS) compared to UCB due to UCB's poor showing on raw classification accuracy (RCA) and *F*-measure. Specifically, the boost in minority class recall for UCB causes a large drop in classification accuracy. While UCB has the best rank for minority class recall and AUC; it drops to rank 7 and 6 for RCA and *F*-measure, respectively. We will see that this is a common trend for UCB on other datasets as well. Note that in an imbalanced data problem, our goal is to improve minority class recall, without sacrificing RCA (primarily determined by majority class accuracy). Second, we find that Learn^{++}.CDS maintains a better rank compared to Learn^{++}.NSE and SEA in many of the measures in Table 2. This demonstrates the value and effectiveness of applying synthetic sampling (SMOTE) to Learn^{++}.NSE for learning unbalanced classes. Third, Learn^{++}.NIE and Learn^{++}.CDS algorithms are highly competitive with one another in terms of combined

figure of merit, the OPM. Fourth, we also observe that while SERA does well on *F*-measure, it performs rather poorly on other figures of merit. Hence, Learn^{++}.NIE (*all implementations*) and Learn^{++}.CDS out-rank SERA in terms of the OPM. Finally, we observe from Fig. 8 that Learn^{++}.NIE (*fm*) out ranks the Learn^{++}.NIE's (*gm*) and (*wrm*) implementations.

6.4.2 Rotating Checkerboard Dataset

Fig. 10 and 11 present the results on the rotating checkerboard dataset. The mean values of all figures of merits used in the evaluation are tabulated in Table 3. We observe several trends, which appear to be common with those observed in the Gaussian experiment. First, Learn^{++}.NSE has very good accuracy (RCA), but primarily due to its performance on the majority class. Yet, Learn^{++}.CDS further improves the OPM by utilizing SMOTE to build more robust classifiers for unbalanced data. In fact, Learn^{++}.CDS performs consistently well across all figures of merit, with the best mean rank, closely followed by Learn^{++}.NIE. Learn^{++}.NIE (*fm*) and Learn^{++}.CDS provide significant improvement to AUC, recall and *F*-measure compared to their predecessor Learn^{++}.NSE. Second, as in the Gaussian dataset, UCB ranks best in terms of minority class recall but ranks lowest in terms of RCA. UCB also experiences a large drop in rank when using the *F*-measure for evaluation. This large drop in rank across measures is not observed for Learn^{++}.NIE (*all implementations*) or Learn^{++}.CDS. Also, observe that Learn^{++}.NIE (*fm*) maintains a significantly better rank than the (*gm*) or (*wrm*) implementation of Learn^{++}.NIE in terms of OPM and other measures, similar to what was observed with the Gaussian problem. Finally, the performance peaks that appear every 50 time steps coincide with the checkerboard being at right angle, corresponding to the simplest distribution for classification.

6.4.3 Shifting Hyperplane

Fig. 12 and 13 present the results on the shifting hyperplane problem. The mean values of all figures of merits used in the evaluation are tabulated in Table 4. As in other experiments, we first note that Learn^{++}.NIE (*all implementations*) significantly outperforms Learn^{++}.NSE and SEA in terms of minority class recall. Because this is an abrupt, sudden concept change problem, rather than a gradual concept drift problem, of primary importance is the speed of recovery after the sudden change, as well as maintaining a high accuracy during the steady state periods. From Fig. 12(d) we observe that Learn^{++}.NIE (*fm*) maintains the best speedy recovery from a sudden change and a high steady state performance in comparison to other implementations of Learn^{++}.NIE. Also, Learn^{++}.NIE (*fm*) does not incur a sudden drop in recall when the hyperplane shifts as compared to other Learn^{++}.NIE implementations. Second, Learn^{++}.CDS and Learn^{++}.NIE (*all implementations*) are the top ranking algorithms in terms of the overall performance measure and mean rank. Third, SERA maintains a constant RCA for nearly all time stamps as shown in Fig. 13(a). This is a result of SERA not using classifiers from previous time stamps, thus there is no prior knowledge retained about the majority class. Finally, we observe, again, that the boost in minority class recall for UCB comes at the cost of the overall accuracy and *F*-measure.

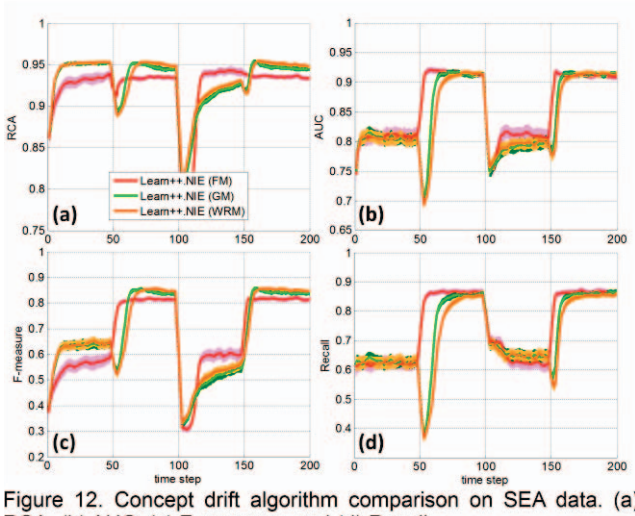

Figure 12. Concept drift algorithm comparison on SEA data. (a) RCA, (b) AUC, (c) F-measure, and (d) Recall

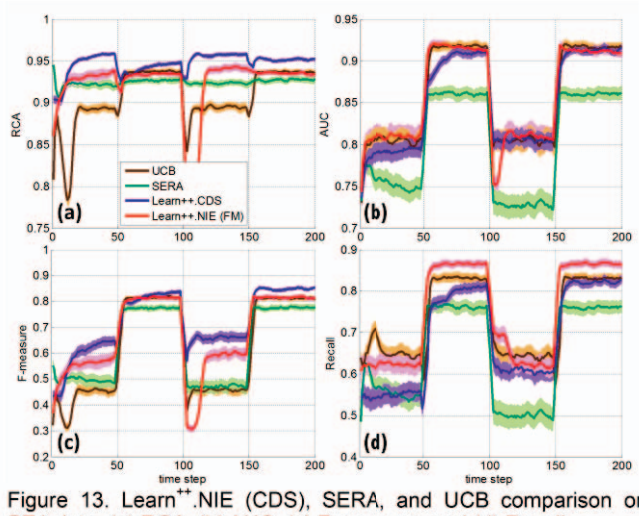

Figure 13. Learn^{++}.NIE (CDS), SERA, and UCB comparison on SEA data. (a) RCA, (b) AUC, (c) F-measure, and (d) Recall

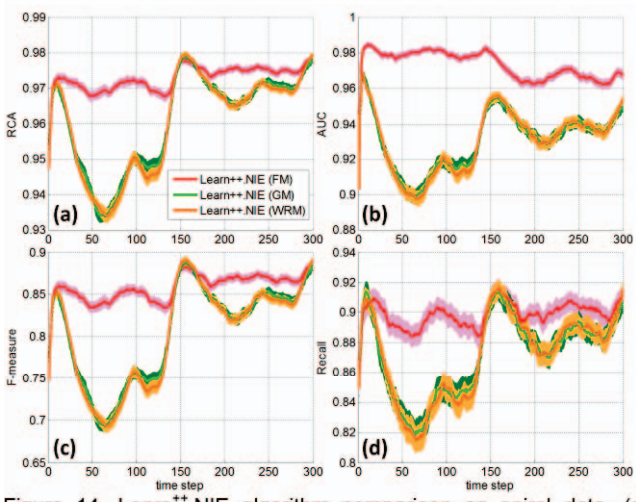

Figure 14. Learn^{++}.NIE algorithm comparison on spiral data. (a) RCA, (b) AUC, (c) F-measure, and (d) Recall

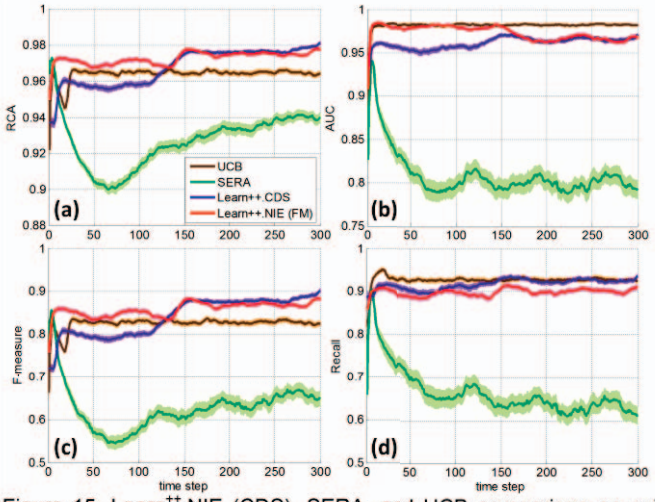

Figure 15. Learn^{++}.NIE (CDS), SERA, and UCB comparison on spiral data. (a) RCA, (b) AUC, (c) F-measure, and (d) Recall

6.4.4 Rotating Spirals Dataset

Fig. 14 and 15 present the results on the rotating spiral dataset. The mean values of all figures of merits used in the evaluation are tabulated in Table 5. One prominent observation on this recurring environment is all implementations of Learn^{++} algorithms are able to use old information stored in the ensemble to improve several figures of merit when the environment reoccurs. This is a custom-designed property of all Learn^{++} family of algorithms, due to their weighting mechanisms that can deactivate and reactivate classifiers based on whether they are relevant to the current environment. Notice that all versions of Learn^{++} algorithms (though some – such as *wrm* implementation - more strongly than others) achieve a significant boost in RCA, minority recall, *F*-measure and AUC when the recurring environment is encountered (time stamps > 150) as shown in Fig. 14. Several other observations are also worth noting. First, while Learn^{++}.NSE has the best raw classification accuracy, again this is due to majority class classification. Lacking a mechanism to accommodate imbalanced data, Learn^{++}.NSE performs poorly on recall and AUC. Following Learn^{++}.NSE, Learn^{++}.NIE (*fm*) has the best raw classification accuracy, and unlike Learn^{++}.NSE, it maintains a high performance on all figures of merit. It is highly competitive with

Table 2. Drifting Gaussians dataset performance and ranking summary

	RCA	F-measure	AUC	Recall	OPM	Average Rank
Learn++.NSE	97.63±0.18 (1)	66.30±2.62 (4)	83.65±1.43 (7)	58.33±3.15 (7)	76.48±1.85 (7)	5.2
SEA	97.46±0.18 (3)	64.39±2.44 (5)	82.97±1.31 (8)	56.40±2.84 (8)	75.31±1.69 (8)	6.4
Learn++.NIE(*fm*)	96.11±0.27 (5)	67.30±1.95 (3)	95.80±0.67 (2)	86.74±2.01 (2)	86.45±0.99 (2)	2.8
Learn++.NIE(*gm*)	95.24±0.27 (6)	63.37±1.86 (7)	92.12±0.89 (4)	86.51±1.90 (3)	84.31±1.23 (4)	4.8
Learn++.NIE(*wrm*)	95.20±0.28 (8)	62.93±1.91 (8)	91.60±0.94 (5)	85.42±1.97 (4)	83.79±1.28 (5)	6.0
Learn++.CDS	97.50±0.20 (2)	74.21±1.90 (1)	92.19±1.07 (3)	80.85±2.45 (5)	86.19±1.41 (3)	2.8
SERA	97.37±0.22 (4)	70.76±2.28 (2)	85.99±1.46 (6)	73.52±2.96 (6)	81.91±1.73 (6)	4.8
UCB	95.22±0.30 (7)	63.74±1.94 (6)	96.84±0.54 (1)	92.02±1.56 (1)	86.96±1.09 (1)	3.2

Table 3. Rotating checkerboard dataset performance and ranking summary

	RCA	F-measure	AUC	Recall	OPM	Average Rank
Learn++.NSE	97.45±0.17 (1)	68.25±2.14 (2)	83.76±1.17 (4)	56.55±2.48 (7)	76.50±1.49 (3)	3.4
SEA	87.41±0.63 (7)	21.93±1.63 (8)	65.75±1.29 (8)	31.87±2.18 (8)	51.74±1.43 (8)	7.8
Learn++.NIE(*fm*)	95.06±0.47 (3)	61.45±2.51 (3)	92.62±0.85 (1)	74.32±2.20 (3)	80.86±1.51 (2)	2.4
Learn++.NIE(*gm*)	90.02±0.51 (5)	42.11±1.94 (5)	83.37±1.13 (5)	66.76±2.20 (5)	70.57±1.45 (6)	5.2
Learn++.NIE(*wrm*)	89.89±0.51 (6)	41.15±1.86 (6)	82.75±1.12 (6)	65.91±2.16 (6)	69.93±1.41 (7)	6.2
Learn++.CDS	97.18±0.21 (2)	72.93±1.82 (1)	90.89±0.96 (3)	74.50±2.19 (2)	83.88±1.30 (1)	1.8
SERA	92.89±0.43 (4)	52.57±2.29 (4)	80.80±1.29 (7)	67.39±2.55 (4)	73.41±1.64 (5)	4.8
UCB	85.78±0.51 (8)	38.26±1.44 (7)	91.89±0.70 (2)	82.33±1.75 (1)	74.57±1.10 (4)	4.4

Table 4. Shifting hyperplane dataset performance and ranking summary

	RCA	F-measure	AUC	Recall	OPM	Average Rank
Learn++.NSE	94.98±0.26 (1)	71.98±1.57 (2)	83.30±0.90 (6)	62.87±1.96 (7)	78.28±1.17 (5)	4.2
SEA	94.00±0.26 (3)	68.13±1.48 (3)	82.00±0.85 (7)	60.28±1.77 (8)	76.10±1.09 (7)	5.6
Learn++.NIE(*fm*)	92.38±0.46 (7)	67.27±1.62 (6)	85.93±0.90 (1)	74.83±1.60 (1)	80.10±1.15 (2)	3.4
Learn++.NIE(*gm*)	93.03±0.31 (5)	67.90±1.36 (5)	84.51±0.81 (4)	72.17±1.61 (3)	79.40±1.02 (3)	4.0
Learn++.NIE(*wrm*)	93.25±0.30 (4)	67.94±1.39 (4)	84.08±0.83 (5)	70.65±1.65 (4)	78.98±1.07 (4)	4.2
Learn++.CDS	94.75±0.28 (2)	72.24±1.46 (1)	85.16±0.84 (3)	68.80±1.79 (5)	80.24±1.09 (1)	2.4
SERA	92.47±0.44 (6)	63.01±1.84 (7)	80.11±1.08 (8)	64.68±2.17 (6)	75.07±1.38 (8)	7.0
UCB	90.77±0.45 (8)	62.05±1.44 (8)	85.84±0.95 (2)	73.34±1.66 (2)	78.00±1.13 (6)	5.2

Table 5. Rotating spirals dataset performance and ranking summary

	RCA	F-measure	AUC	Recall	OPM	Average Rank
Learn++.NSE	97.76±0.11 (1)	86.13±0.76 (1)	91.33±0.49 (6)	76.96±1.17 (6)	88.05±0.63 (6)	4.0
SEA	96.65±0.12 (4)	78.97±0.84 (7)	88.91±0.50 (7)	69.49±1.15 (7)	83.51±0.65 (7)	6.4
Learn++.NIE(*fm*)	97.30±0.13 (2)	85.87±0.65 (2)	97.34±0.26 (2)	89.87±0.73 (3)	92.60±0.44 (1)	2.0
Learn++.NIE(*gm*)	96.11±0.16 (6)	80.57±0.70 (5)	93.11±0.38 (4)	87.21±0.80 (4)	89.25±0.51 (4)	4.6
Learn++.NIE(*wrm*)	96.08±0.16 (7)	80.46±0.70 (6)	93.09±0.39 (5)	87.20±0.80 (5)	89.21±0.51 (5)	5.6
Learn++.CDS	96.81±0.15 (3)	84.15±0.65 (3)	96.15±0.31 (3)	91.77±0.71 (2)	92.22±0.46 (3)	2.8
SERA	92.73±0.32 (8)	62.67±1.66 (8)	80.96±1.10 (8)	66.57±2.17 (8)	75.73±1.45 (8)	8.0
UCB	96.42±0.16 (5)	82.57±0.69 (4)	98.18±0.19 (1)	92.74±0.65 (1)	92.48±0.42 (2)	2.6

other algorithms across all measures. In fact, Learn^{++}.NIE (*fm*) outperforms all other algorithms in overall performance (OPM), and has the best mean rank across all measures among all algorithms, including Learn^{++}.NIE with *wrm* and *gm*. This was observed on previous datasets as well. We also observe a significant increase in OPM and mean rank for Learn^{++}.CDS (which adds SMOTE to Learn^{++}.NSE) over its predecessor Learn^{++}.NSE, demonstrating the impact of adding SMOTE to Learn^{++}.NSE for learning unbalanced classes.

Table 6. Electricity pricing dataset performance and ranking summary

	RCA	F-measure	AUC	Recall	OPM	Average Rank
Learn++.NSE	90.75±0.86 (2)	15.40±3.05 (7)	59.66±2.04 (7)	16.87±3.31 (7)	45.67±2.32 (7)	6.0
SEA	92.15±0.60 (1)	9.37±2.15 (8)	58.48±1.55 (8)	10.53±2.19 (8)	42.63±1.62 (8)	6.6
Learn++.NIE(*fm*)	82.60±1.80 (6)	20.79±2.55 (3)	72.45±2.15 (1)	38.72±4.93 (3)	53.64±2.86 (3)	3.2
Learn++.NIE(*gm*)	83.60±1.30 (5)	22.29±2.64 (1)	70.70±2.34 (2)	38.37±4.68 (4)	53.74±2.74 (2)	2.8
Learn++.NIE(*wrm*)	84.70±1.15 (4)	21.88±2.61 (2)	69.54±2.23 (4)	35.61±4.28 (5)	52.93±2.57 (4)	3.8
Learn++.CDS	88.48±1.12 (3)	18.09±3.05 (6)	60.58±2.27 (6)	22.91±4.07 (6)	47.52±2.63 (6)	5.4
SERA	76.42±1.70 (7)	19.91±2.06 (4)	62.42±2.22 (5)	46.46±4.70 (2)	51.30±2.67 (5)	4.6
UCB	68.23±1.72 (8)	18.68±1.75 (5)	69.74±2.34 (3)	58.87±4.47 (1)	53.88±2.57 (1)	3.6

Table 7. NOAA weather prediction dataset performance and ranking summary

	RCA	F-measure	AUC	Recall	OPM	Average Rank
Learn++.NSE	73.35±0.00 (4)	51.27±0.00 (5)	72.08±0.00 (6)	49.38±0.00 (6)	61.52±0.00 (5)	5.2
SEA	75.81±0.00 (1)	50.43±0.00 (6)	73.37±0.00 (4)	42.86±0.00 (8)	60.62±0.00 (6)	5.0
Learn++.NIE(*fm*)	70.54±1.08 (7)	59.19±1.31 (3)	77.84±0.79 (1)	72.48±2.19 (1)	70.01±1.34 (2)	2.8
Learn++.NIE(*gm*)	73.53±0.80 (3)	60.78±1.12 (2)	76.83±0.69 (2)	69.27±1.84 (2)	70.10±1.11 (1)	2.0
Learn++.NIE(*wrm*)	74.07±0.74 (2)	60.94±1.04 (1)	76.42±0.66 (3)	68.04±1.71 (3)	69.87±1.04 (3)	2.4
Learn++.CDS	73.05±0.93 (5)	52.89±1.74 (4)	72.91±1.03 (5)	53.75±2.69 (5)	63.15±1.60 (4)	4.6
SERA	65.17±1.83 (8)	48.38±2.30 (7)	63.54±1.48 (8)	58.49±4.16 (4)	58.90±2.44 (7)	6.8
UCB	70.82±1.43 (6)	46.40±3.18 (8)	71.07±1.57 (7)	45.54±4.77 (7)	58.46±2.74 (8)	7.2

Finally, this dataset identifies a specific weakness of the SERA algorithm, which performs particularly poorly with the worst rank in every measure. A unique feature of this dataset is that the mean as well as the covariance of the classes remain unchanged despite the drift, and they are equal for all classes. Yet, SERA's selection of minority class instances depends on a (Mahalonobis-based) similarity measure, which remains unchanged despite the drift. All Learn^{++} algorithms, as well as UCB, are immune to this problem, as they are not based on a distance metric.

6.4.5 Electricity Pricing Dataset

Fig. 16 and 17 present the results on the electricity pricing dataset, a particularly challenging problem once it has been converted to an imbalanced learning problem. This can be seen from Table 6, showing low overall figures in *F*-measure, recall and OPM for all algorithms tested. A few additional observations: first, observe that UCB has the best overall performance measure, followed by Learn^{++}.NIE. However, Learn^{++}.NIE (*fm*) and (*gm*) implementations have the highest mean rankings. Furthermore, as in previous datasets, UCB's strong recall performance comes at the cost of very poor RCA.

6.4.6 NOAA Weather Prediction

Fig. 18 and 19 present the results on the NOAA weather dataset. The mean values of all figures of merits used in the evaluation are tabulated in Table 7. Unlike previous datasets, Table 7 does not indicate any variation in Learn^{++}.NSE or SEA results because the generated base classifier (CART algorithm) is identical for each trial when we generate a classifier on any fixed batch of data. As in previous datasets, however, we do observe better rankings for Learn^{++}.NIE algorithms. The differences between the Learn^{++}.NIE implementations were not statistically significant, whereas the improvement of any of the Learn^{++}.NIE implementations over any of the other algorithms were all significant, and often by wide margins.

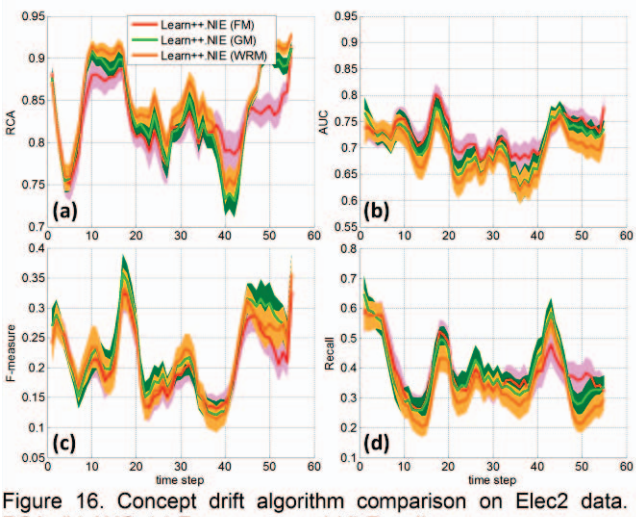

Figure 16. Concept drift algorithm comparison on Elec2 data. (a) RCA, (b) AUC, (c) F-measure, and (d) Recall

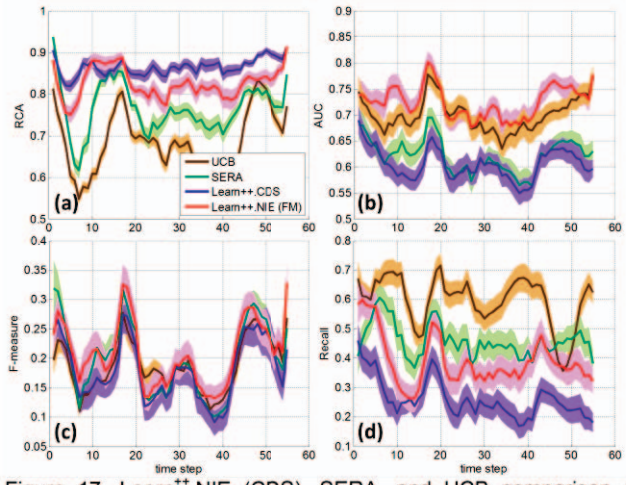

Figure 17. Learn^{++}.NIE (CDS), SERA, and UCB comparison on Elec2 data. (a) RCA, (b) AUC, (c) F-measure, and (d) Recall

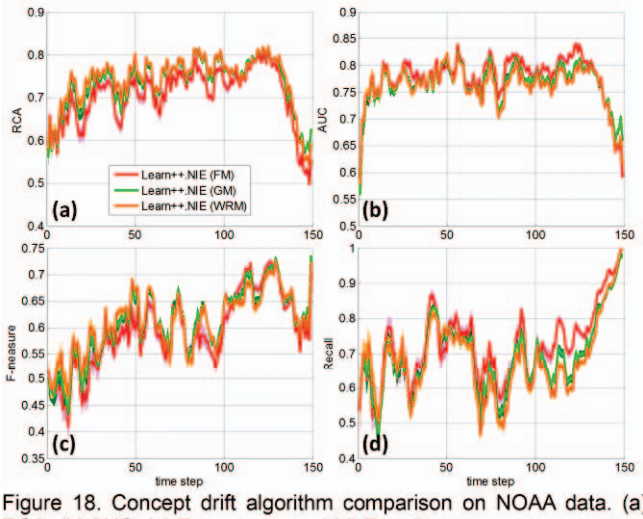

Figure 18. Concept drift algorithm comparison on NOAA data. (a) RCA, (b) AUC, (c) F-measure, and (d) Recall

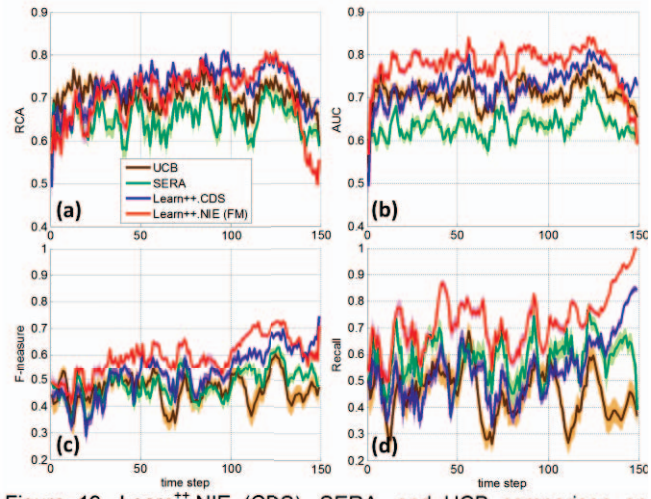

Figure 19. Learn^{++}.NIE (CDS), SERA, and UCB comparison on NOAA data. (a) RCA, (b) AUC, (c) F-measure, and (d) Recall

The only exception to this was again with the raw classification accuracy, where SEA performed the best, which, naturally, is merely due to its majority class performance. Of course, SEA cannot accommodate imbalanced data, so its performances on all other metrics were – as expected – very poor. On the other hand, the three Learn^{++}.NIE implementations shared the top three spots in *F*-measure, AUC, recall, and OPM as well as mean rank. We also notice that both UCB and SERA perform rather poorly across all figures of merit on this dataset.

6.5 Summary of Results

Table 8 provides a summary of overall performance measure (OPM) ranks of each algorithm evaluated on all datasets along with the mean OPM rank averaged over all experiments. This table shows that Learn^{++}.NIE (*fm*), Learn^{++}.CDS and Learn^{++}.NIE (*gm*) occupy the top three spots in OPM as well as the mean rank, respectively. This table appears to indicate that these Learn^{++} algorithms outperform others. The table does not tell however, whether we can claim – with statistical significance – that any of the algorithms is actually better than the others.

When different algorithms provide varying performances on different datasets, one way to rigorously assess and compare the algorithms is Friedman's test [64], which provides a rank based statistical significance test. This test, a non-parametric alternative to repeated measures ANOVA, tests whether there is a statistical difference among the ranks of different algorithms. The test, when run over all algorithms, shows that the null hypothesis for each measure – that the rankings of the algorithms are equivalent – is rejected, indicating that at least one algorithm is better than the others on each metric. Again, following the test description in [64], we compute a z-score as a test statistic for pair wise comparison of algorithm i vs. algorithm j ($i \neq j$), where $R_m(j)$ is the average rank of algorithm j on measure m, k is the number of algorithms under test and N is the number of datasets used.

$$z_m(i,j) = \frac{R_m(i) - R_m(j)}{\sqrt{\frac{k(k+1)}{6N}}} \tag{22}$$

Since Table 8 lists Learn^{++}.NIE (*fm*) and Learn^{++}.CDS as the top ranking algorithms, we compare each of these two algorithms to all others on all measures – based on the pair wise Friedman test – to determine whether the performance increase – if any – is significant. The results are provided in Table 9. Significance is indicated by • for Learn^{++}.NIE (*fm*) and by ■ for Learn^{++}.CDS. We also apply the Bonferroni-Dunn correction to account for multiple comparisons of a group of algorithms to a control classifier (e.g., either Learn^{++}.NIE(*fm*) or Learn^{++}.CDS). We separate the Bonferroni-Dunn test into two groups, namely concept drift algorithms (baseline vs. SEA/Learn^{++}.NSE) and concept drift/class imbalance algorithms (baseline vs. Learn^{++}.NIE/CDS/SERA/UCB). We note that without this conservative correction, the results showed more significant differences in favor of Learn^{++}.NIE and/or Learn^{++}.CDS even in the boxes that do not currently show significance. We observe that Learn^{++}.NIE (*fm*) and Learn^{++}.CDS are both significantly better than Learn^{++}.NSE and SEA on AUC, recall and OPM, but we cannot not claim significance on RCA. This is not surprising, as Learn^{++}.NSE and SEA are designed for concept drift problems, and they perform very well on such problems, where their RCA performance is primarily due to their accuracy on the majority class. Of course, we note that raw classification accuracy, alone, is never a good figure of merit on an imbalanced dataset. One or both of Learn^{++}.NIE(*fm*) and Learn^{++}.CDS also outperform SERA on all measures except *F*-measure. Learn^{++}.CDS significantly outperforms UCB in raw classification accuracy as well as *F*-measure (which itself is a combined measure of recall and precision), whereas Learn^{++}.NIE(*fm*) significantly outperforms UCB on *F*-measure only. We emphasize that both Learn^{++}.NIE(*fm*) and Learn^{++}.CDS do outperform UCB on other measures, when averaged over all datasets, but there is not sufficient evidence to determine the improvement is statistically significant. Learn^{++}.NIE(*fm*) and UCB tie in terms of mean rank for recall across all datasets. Learn^{++}.NIE(*fm*) has the lowest (i.e., best) mean rank in AUC compared to UCB; however, there is not sufficient evidence to claim significance.

Table 8. Summary of OPM ranks of all the algorithms on all datasets

	Gauss	Checker	Spiral	Hyperplane	Elec	NOAA	Average	Final Rank
Learn^{++}.NSE	7	3	5	6	7	5	5.50	6
SEA	8	8	7	7	8	6	7.33	8
Learn^{++}.NIE(*fm*)	2	2	2	1	3	2	2.00	1
Learn^{++}.NIE(*gm*)	4	6	3	4	2	1	3.33	3
Learn^{++}.NIE(*wrm*)	5	7	4	5	4	3	4.67	5
Learn^{++}.CDS	3	1	1	3	6	4	3.00	2
SERA	6	5	8	8	5	7	6.50	7
UCB	1	4	6	2	1	8	3.67	4

Table 9. Hypothesis testing comparing Learn^{++}.NIE (*fm*) [•] and Learn^{++}.CDS [■] to other algorithms used during the presentation of results (only significant improvement is marked)

	Learn^{++}.NSE	SEA	SERA	UCB
RCA			■	■
F-measure		•■		•■
AUC	•■	•■	•■	
Recall	•■	•■	•	
OPM	•■	•■	•■	

It is important to remember that in an imbalanced data environment, we seek a classifier that provides the best overall balance in classification accuracy, recall, precision and AUC. While no algorithm has absolute superiority on all others on all figures of merit, we see that Learn^{++}.NIE (*fm*) and Learn^{++}.CDS outperform other algorithms far more often and with significance then their competitors when tested on a variety of synthetic and real world datasets that cover a broad spectrum of nonstationary environments.

7 Discussion & Conclusions

In this paper, we discuss learning concept drift from imbalanced data. While each of these two areas has been well researched, the joint problem – despite its growing prevalence in real world – is mostly underexplored. We reviewed the relevant literature, and in particular, the UCB and SERA algorithms as the seminal approaches proposed for the joint problem, discussing their strength and weaknesses. We propose two ensemble based approaches that can learn in a wide spectrum of concept drift scenarios that feature heavy class imbalance, while avoiding the primary bottleneck of the existing approaches, namely, accumulating minority data and using partial old data. The proposed approaches, Learn^{++}.NIE and Learn^{++}.CDS, are truly incremental approaches that do not require access to previous data, and they do not accumulate minority data to balance the class cardinalities. Instead, Learn^{++}.CDS uses SMOTE to rebalance the classes, whereas Learn^{++}.NIE uses sub-ensembles with bagging, along with alternate error metrics to learn from imbalanced data. Both algorithms have the ability to learn new knowledge and preserve prior knowledge about the environment, which is particularly useful for recurring concepts as we observed with the spiral dataset.

Specifically, Learn^{++}.CDS is a natural integration of the two algorithms, Learn^{++}.NSE [7] and SMOTE [46], previously established approaches for concept drift and imbalanced data problems, respectively. This straightforward combination is

very robust when analyzed on several synthetic and real-world datasets. The idea behind this approach was to use SMOTE to rebalance the classes using synthetic minority class data, followed by using Learn^{++}.NSE on rebalanced data.

Learn^{++}.NIE, however, uses a different strategy that is based on two pillars: 1) use bagging based sub-ensembles to reduce class imbalance (without generating synthetic data, and without accumulating minority data); and 2) use different measures that focus on both class-specific performances to weigh classifiers. We have shown that using measures other than a class independent error, such as the weighted recall, *F*-measure, or geometric mean of the individual class performances, can be very effective in tracking concept drift, boost the performance on minority class, all the while avoid catastrophic performance drop in majority class classification. The classifier weighting measures presented in this paper were selected to reward classifiers in the ensemble that perform well on *all* classes rather than an error measure that may be biased towards a majority class or do not examine the error on a minority class.

We compared the proposed approaches to other algorithms, UCB and SERA, both specifically designed for class imbalance and concept drift. Both algorithms retain and accumulate all (as in UCB) or part (as in SERA) of the minority class data at each time stamp. During the experimentation process, we kept all algorithm parameters constant in order to maintain a fair comparison (also using the free parameter values as suggested by their authors in the respective original work).

The performance gains of Learn^{++} algorithms over others do come at a cost, however: computational complexity. In particular Learn^{++}.NIE, the best overall performing algorithm, is also the computationally most expensive one. This is not surprising, since Learn^{++}.NIE generates an ensemble of ensembles, due to the sub-ensemble created at each time step. For UCB and SERA algorithms, the bottleneck is with accumulating the minority data, which need to be stored for training future classifiers. Learn^{++} algorithms do not store old data, as they are truly incremental algorithms; however, building sub-ensembles at each time stamp is computationally more expensive than accumulating minority data. Furthermore, UCB and SERA maintain a fixed ensemble size, whereas Learn^{++} algorithms do not force a fixed ensemble size. If fixed ensemble size is desired, pruning methods can be implemented as discussed by Elwell & Polikar for Learn^{++}.NSE in [39]. As a result, when comparing the algorithms with respect to their computational complexities, we observed that UCB is the fastest, followed by SEA and SERA, and then by Learn^{++}.NSE, Learn^{++}.CDS, and finally Learn^{++}.NIE.

We should note that the complexity added for all Learn^{++} algorithms, as well as those of others, is linear in the number of classifiers, and since a fixed number of classifiers are generated per dataset (the subensemble size for Learn^{++}.NIE, and 1 for all others) , it is also linear in the number of data batches. However, the actual complexity of each algorithm, of course, depends on the mechanism used to rebalance the data as well as the complexity of the base classifier used. Figure 20 shows the timing diagrams of two experiments: the one with the largest number of classifiers (spiral dataset, ~300 time

steps) as well as that of real-world weather dataset (~150 time steps). The diagrams for all experiments were very similar. The vertical axis is the total training time (rebalancing, training the new classifier, computing the weights and evaluating the classifiers) for each of the algorithm at each time step (hence for each added classifier). With the exception of minor variations (due to underlying rebalancing mechanism, and/or the base classifier), all algorithm's evaluation time increases in a linear fashion. Even for the spiral dataset, where we had the largest number of classifiers, the time it took to train the final classifier and evaluate all ~300 classifiers was about 1.2 seconds on a very modestly configured laptop with AMD Phenom II X6 processor with 8Gb of memory running Ubuntu Linux 11.10. We should add, of course, the computational complexity described here is the worst case and most conservative scenario for the Learn^{++} algorithms: if we were to limit and fix the ensemble size (as it is done with SEA, SERA and UCB), Learn^{++} algorithms would dramatically reduce their computational complexities, and become quite competitive with others.

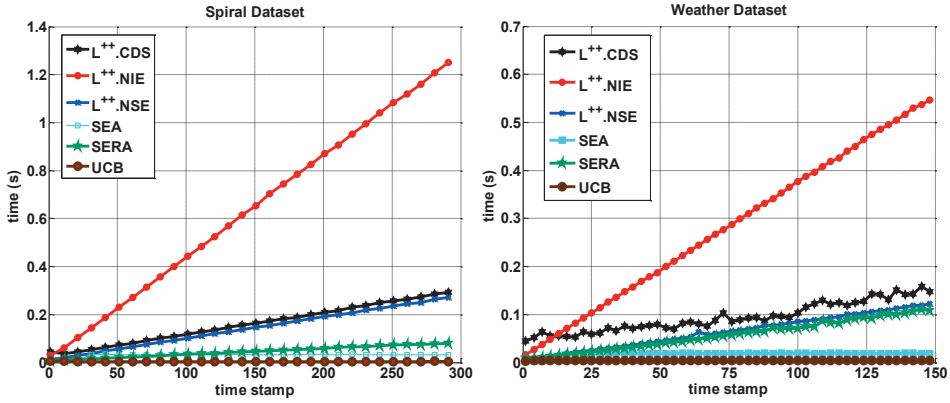

Figure 20. Timing diagrams for spiral and weather datasets

A reasonable question that now needs to be answered is which algorithm to use and when. In an imbalanced data concept drift scenario, we have several criteria and constraints that are sometimes conflicting in their nature: of course, we want good classification accuracy in general, but we also want to be able to recall the minority class data, maintain good performance on majority data, and maintain a healthy AUC. We observed, when averaged across multiple datasets and figures of merits, Learn^{++} based approaches typically provided a better-balanced performance, most with statistical significance. Based on our observations on several datasets and figures of merit, we reach the following set of guidelines:

- *Learn^{++}.NSE*: Use Learn^{++}.NSE when there is concept drift in the data and the classes are relatively balanced. If classes are imbalanced, Learn^{++}.NSE may still obtain a relatively good classification accuracy – a potentially misleading result – as that will be based on its majority class performance. The recall of minority class will suffer.
- *UCB*: This algorithm, came fourth in our overall ranking, is most suitable if the minority class does *not* drift, and it is the minority class recall that is the most important figure of merit. UCB generally provided the best minority recall performance, though at the cost of classification accuracy of the majority class. On the other hand, this is com-

putationally the fastest algorithm, making it most suitable for applications where new data are continuously generated in very rapid succession.

- *SERA*: Also suited for problems where minority data do not drift, with overall performance better than that of UCB. SERA is not as prone to drifting minority class as much as UCB. Both UCB and SERA are not strictly incremental, hence are better suited if previously seen data can be retained and used in the future.
- *Learn^{++}.CDS*: Use Learn^{++}.CDS if both minority and majority concepts are drifting, classes contain imbalance and memory and computational complexity considerations are critical. Learn^{++}.CDS has a smaller memory requirement than Learn^{++}.NIE, as it does not need to generate sub-ensembles. Both Learn^{++} algorithms satisfy strict incremental learning criteria and do not need access to any prior data.
- *Learn^{++}.NIE*: Learn^{++}.NIE is the better overall algorithm, if both minority and majority concepts are drifting and a strong balanced performance is needed on both minority and majority classes. Learn^{++}.NIE uses a weight that reflects the performance on weighted recall measure (*wrm*), *F*-measure, or *G*-mean. The *F*-measure weighting scheme typically provided the best results on a broad array of learning scenarios. Also, while Learn^{++}.NIE with *wrm* came third in overall ranking, it has the distinct feature to control performance for class specific recall through its η parameter (see [60]). Learn^{++}.NIE is computationally the most expensive, though it appears that it is still quite fast – using standard computational power typically available today – to accommodate just about most streaming applications. Both Learn^{++}.CDS and Learn^{++}.NIE can be made much faster by fixing ensemble size.

While this effort demonstrated the empirical proof-of-concept of the Learn^{++}.CDS and Learn^{++}.NIE algorithms, we would like to explore whether any theoretical performance guarantees can be offered by these algorithms. The infinitely many variations of nonstationary environments that one can encounter make such an analysis a difficult proposition. However, formal statistical analyses of these algorithms, at least on certain nonstationary environments – such as Gaussian distribution drifts – are within the scope of our current and future work.

ACKNOWLEDGMENTS

This material is based on the work supported by the National Science Foundation (NSF) under Grant No: ECCS-092159.

REFERENCES

[1] C. Giraud-Carrier, "A note on the utility of incremental learning," *Artificial Intelligence Communications*, vol. 13, no. 4, pp. 215-223, 2000.
[2] S. Lange and S. Zilles, "Formal models of incremental learning and their analysis," *Int. Joint Conf. on Neural Networks,* vol. 4, pp. 2691-2696, 2003.
[3] M. D. 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 Transactions on Neural Networks*, vol. 20, no. 1, pp. 152-168, 2009.
[4] L. I. Kuncheva, "Classifier Ensembles for Changing Environments," *Multiple Classifier Systems*, pp. 1-15, 2004.
[5] A. Bifet, "Adaptive Learning and Mining for Data Streams and Frequent Patterns." Ph.D. Dissertation, Universitat Politècnica de Catalunya, 2009.
[6] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural Netw.*, vol. 1, no. 1, pp. 17-61, 1988.
[7] R. Elwell and R. Polikar, "Incremental Learning of Concept Drift in Nonstationary Environments," *IEEE Transactions on Neural Networks*, vol. 22, no.

10, pp. 1517-1531, Oct.2011.

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

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

[10] 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.

[11] J. Case, S. Jain, S. Kaufmann, A. Sharma, and F. Stephan, "Predictive learning models for concept drift," *Theoretical Computer Science*, vol. 268, no. 2, pp. 323-349, Oct.2001.

[12] F. H. Hamker, "Life-long learning Cell Structures--continuously learning without catastrophic interference," *Neural Networks*, vol. 14, no. 4-5, pp. 551-573, May2001.

[13] C. Alippi and M. Roveri, "Just-in-Time Adaptive Classifiers - Part I: Detecting Nonstationary Changes," *IEEE Trans. Neural Networks*, vol. 19, no. 7, pp. 1145-1153, 2008.

[14] C. Alippi and M. Roveri, "Just-in-Time Adaptive Classifiers - Part II: Designing the Classifier," *IEEE Transactions on Neural Networks*, vol. 19, no. 12, pp. 2053-2064, Dec.2008.

[15] C. Alippi, G. Boracchi, and M. Roveri, "Just in time classifiers: managing the slow drift case," *International Joint Conference on Neural Networks (IJCNN 2009)*, pp. 114-120, Atlanta, GA, 2009.

[16] C. Alippi, G. Boracchi, and M. Roveri, "Change Detection Tests Using the ICI Rule," *World Congress on Computational Intelligence (WCCI 2010) - International Joint Conference on Neural Networks (IJCNN 2010)*, pp. 1190-1196, Barcelona, Spain, 2010.

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

[18] S. Hoeglinger and R. Pears, "Use of Hoeffding trees in concept based data stream mining," *International Conference on Information and Automation for Sustainability (CIAFS 2007)*, pp. 57-62, 2007.

[19] C. J. Tsai, C. I. Lee, and W. P. Yang, "Mining decision rules on data streams in the presence of concept drifts," *Expert Systems with Applications*, vol. 36, no. 2, Part 1, pp. 1164-1178, Mar.2009.

[20] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," *Conf. on Knowledge Discovery in Data*, pp. 97-106, San Francisco, 2001.

[21] L. Cohen, G. Avrahami, M. Last, and A. Kandel, "Info-fuzzy algorithms for mining dynamic data streams," *Applied Soft Computing*, vol. 8, no. 4, pp. 1283-1294, Sept.2008.

[22] 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*, vol. 9, no. 3, pp. 344-353, 2008.

[23] M. Baena-Garcia, J. del Campo-Avila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Bueno-Morales, "Early drift detection method," *ECML PKDD Workshop on Knowledge Discovery from Data Streams*, pp. 77-86, 2006.

[24] L. I. Kuncheva, "Classifier ensembles for detecting concept change in streaming data: overview and perspectives," *European Conference on Artificial Intelligence (ECAI)*, pp. 5-10, Patras, Greece, 2008.

[25] W. N. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," *Int'l Conf. on Knowledge Discovery & Data Mining*, pp. 377-382, 2001.

[26] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavalda, "New Ensemble Methods For Evolving Data Streams," *15th ACM SIGKDD international conference on knowledge discovery and data mining (KDD 09)*, pp. 139-148, 2009.

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

[28] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: an ensemble method for drifting concepts," *Journal of Machine Learning Research*, vol. 8, pp. 2755-2790, 2007.

[29] J. Gao, W. Fan, and J. Han, "On appropriate assumptions to mine data streams: analysis and practice," *International Conference on Data Mining (ICDM 2007)*, pp. 143-152, 2007.

[30] K. Nishida and K. Yamauchi, "Adaptive Classifiers-Ensemble System for Tracking Concept Drift," *International Conf.on Machine Learning and Cybernetics*, eds. K. Yamauchi, Ed., vol. 6, pp. 3607-3612, 2007.

[31] H. He and S. Chen, "IMORL: Incremental Multiple-Object Recognition and Localization," *IEEE Transactions on Neural Networks*, vol. 19, no. 10, pp. 1727-1738, 2008.

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

[33] H. Abdulsalam, D. Skillicorn, and P. Martin, "Classification Using Streaming Random Forests," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 1, pp. 22-36, 2011.

[34] A. Bifet, E. Frank, G. Holmes, and B. Pfahringer, "Accurate ensembles for data streams: Combining restricted Hoeffding trees using stacking.," *2nd Asian Conference on Machine Learning* in Journal of Machine Learning Research, vol. 13, Tokyo, 2010.

[35] A. Bifet, MOA: Massive Online Analysis, Available at: http://moa.cs.waikato.ac.nz/. Last accessed:7/6/2011.

[36] T. R. Hoens, N. V. Chawla, and R. Polikar, "Heuristic Updatable Weighted Random Subspaces for Non-stationary Environments," *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pp. 241-250, 2011.

[37] M. Muhlbaier and R. Polikar, "An Ensemble Approach for Incremental Learning in Nonstationary Environments," *Multiple Classifier Systems*, pp. 490-500, 2007.

[38] M. Karnick, M. Ahiskali, M. Muhlbaier, and R. Polikar, "Learning concept drift in nonstationary environments using an ensemble of classifiers based approach," *Int'l Joint Conf.on Neural Netw.*, pp. 3455-3462, 2008.

[39] R. Elwell, R. Polikar, "Incremental Learning in Nonstationary Environments with Controlled Forgetting," *International Joint Conference on Neural Net-*

works (IJCNN 2009), pp. 771-778, 2009.

[40] R. Elwell and R. Polikar, "Incremental learning of variable rate concept drift," *International Workshop on Multiple Classifier Systems (MCS 2009)* in Lecture Notes in Computer Science, vol. 5519, pp. 142-151, Reykjavik, Iceland, 2009.

[41] R. Polikar, L. Upda, S. S. Upda, and V. Honavar, "Learn++: an incremental learning algorithm for supervised neural networks," *IEEE Trans. Syst. , Man, Cybern. A, Syst. , Humans*, vol. 31, no. 4, pp. 497-508, 2001.

[42] M. Kubat, R. Holte, and S. Matwin, "Machine Learning for the Detection of Oil Spills in Satellite Radar Images ," *Machine Learning*, vol. 30, pp. 195-215, 1998.

[43] H. He and E. A. Garcia, "Learning from Imbalanced Data," *IEEE Tran. on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263-1284, Sept.2009.

[44] P. E. Hart, "The condensed nearest neighbor rule," *IEEE Transactions on Information Theory*, vol. 14, no. 3, pp. 515-516, 1968.

[45] I. Tomek, "Two Modifications of CNN," *IEEE Transactions on Systems, Man and Cybernetics, Part A*, vol. 6, no. 11, pp. 769-772, 1976.

[46] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Oversampling Technique," *Journal of Artificial Intelligence*, vol. 16, pp. 321-357, 2002.

[47] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, " SMOTEBoost: Improving Prediction of the Minority Class in Boosting," *7th European Conference on Principles and Practice of Knowledge Discoveryin Databases (PKDD)*, pp. 107-119, Dubrovnik, Croatia, 2003.

[48] C. Li, "Classifying imbalanced data using a bagging ensemble variation (BEV)," *ACM Southeast Regional Conf.*, pp. 203-208, Winston-Salem, NC, 2007.

[49] G. Ditzler, M. Muhlbaier, and R. Polikar, "Incremental Learning of New Classes in Unbalanced Datasets: Learn++.UDNC," *Int. Workshop on Multiple Classifier Systems (MCS 2010)* in Lecture Notes in Computer Science, vol. 5997, pp. 33-42, 2010.

[50] H. Guo and H. Viktor, "Learning form Imbalanced Data sets with Boosting and Data Generation: The DataBoost-IM Approach," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 30-39, 2004.

[51] T. Yuchun, Z. Yan-Qing, N. V. Chawla, and S. Krasser, "SVMs Modeling for Highly Imbalanced Classification," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 39, no. 1, pp. 281-288, Feb.2009.

[52] R. Batuwita and V. Palade, "FSVM-CIL: Fuzzy Support Vector Machines for Class Imbalance Learning," *Fuzzy Systems, IEEE Transactions on*, vol. 18, no. 3, pp. 558-571, June2010.

[53] J. Gao, W. Fan, J. Han, and P. S. Yu, "A general framework for mining concept-drifting data streams with skewed distributions," *SIAM International Conference on Data Mining*, vol. 7, 2007.

[54] S. Chen and H. He, "SERA: Selectively recursive approach towards nonstationary imbalanced stream data mining," *International Joint Conference on Neural Networks (IJCNN 2009)*, pp. 522-529, Atlanta, GA, 2009.

[55] S. Chen, H. He, L. Kang, and S. Desai, "MuSeRA: Multiple Selectively Recursive Approach towards imbalanced stream data mining," *World Congress on Computer Intelligence (WCCI 2010) - International Joint Conference on Neural Networks (IJCNN 2010)*, pp. 1-8, Barcelona, Spain, 2010.

[56] S. Chen and H. He, "Towards incremental learning of nonstationary imbalanced data stream: a multiple selectively recursive approach," *Evolving Systems*, vol. 2, no. 1, pp. 35-50, 2011.

[57] E. S. Xioufis, M. Spiliopoulou, G. Tsoumakas, and I. Vlahavas, "Dealing with concept drift and class imbalance in multi-label stream classification," *International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pp. 1583-1588, 2011.

[58] G. Ditzler, R. Polikar, and N. Chawla, "An Incremental Learning Algorithm for Non-stationary Environments and Class Imbalance," *20th International Conference on Pattern Recognition (ICPR 2010)*, pp. 2997-3000, Istanbul, Turkey, 2010.

[59] 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.

[60] G. Ditzler and R. Polikar, "An ensemble based incremental learning framework for concept drift and class imbalance," *World Congress on Computational Intelligence (WCCI 2010) - International Joint Conference on Neural Networks (IJCNN 2010)*, pp. 1-8, Barcelona, Spain, 2010.

[61] R. Polikar and G. Ditzler, Benchmark datasets for evaluating concept drift / imbalanced data algorithms, Available at: http://users.rowan.edu/~polikar/research/NIE_data. Last accessed:4/24/2012.

[62] M.B. Harries, SPLICE-2 comparative evaluation: electricity pricing, Available at: ftp://ftp.cse.unsw.edu.au/pub/doc/papers/UNSW/9905.pdf. Last accessed:4/24/2012.

[63] U.S.National Oceanic and Atmospheric Administration (NOAA), Federal Climate Complex Global Surface Summary of Day Data - Version 7 - USAF Datsav3 Station # 725540, Available at: ftp://ftp.ncdc.noaa.gov/pub/data/gsod/ . Last accessed:4/23/2012.

[64] J. Demsar, "Statistical Comparisons of Classifiers over Multiple Datasets," *Journal of Machine Learning Research*, vol. 7, pp. 1-30, 2006.

Gregory Ditzler received the B.S. in electronics engineering technology from the Pennsylvania College of Technology in 2008, and the M.S. in electrical and computer engineering from Rowan University in 2011. He is currently pursuing a Ph.D. in electrical and computer engineering at Drexel University. He was awarded the Graduate Research Achievement Award in 2011 from Rowan University. His current research interests include concept drift, online and incremental learning, multiple classifier systems, and applications of machine learning in bioinformatics

Robi Polikar is a Professor of Electrical and Computer Engineering at Rowan University, in Glassboro, NJ. He has received his B.Sc. degree in electronics and communications engineering from Istanbul Technical University in 1993, and his M.Sc and Ph.D. degrees, both co-majors in electrical engineering and biomedical engineering, from Iowa State University, Ames, IA in 1995 and 2000, respectively. His current research interests within computational intelligence include ensemble systems, incremental and nonstationary learning, and various applications of pattern recognition in bioinformatics and biomedical engineering. He is a member of IEEE, ASEE, Tau Beta Pi and Eta Kappa Nu. His recent and current works are funded primarily through NSF's CAREER and Energy, Power and Adaptive Systems (EPAS) programs.