COMPOSE: A Semisupervised Learning Framework for Initially Labeled Nonstationary Streaming Data

Karl B. Dyer, Robert Capo, and Robi Polikar, *Senior Member, IEEE*

***Abstract*— An increasing number of real-world applications are associated with streaming data drawn from drifting and nonstationary distributions that change over time. These applications demand new algorithms that can learn and adapt to such changes, also known as concept drift. Proper characterization of such data with existing approaches typically requires substantial amount of labeled instances, which may be difficult, expensive, or even impractical to obtain. In this paper, we introduce compacted object sample extraction (COMPOSE), a computational geometry-based framework to learn from nonstationary streaming data, where labels are unavailable (or presented very sporadically) after initialization. We introduce the algorithm in detail, and discuss its results and performances on several synthetic and real-world data sets, which demonstrate the ability of the algorithm to learn under several different scenarios of initially labeled streaming environments. On carefully designed synthetic data sets, we compare the performance of COMPOSE against the optimal Bayes classifier, as well as the arbitrary subpopulation tracker algorithm, which addresses a similar environment referred to as extreme verification latency. Furthermore, using the real-world National Oceanic and Atmospheric Administration weather data set, we demonstrate that COMPOSE is competitive even with a well-established and fully supervised nonstationary learning algorithm that receives labeled data in every batch.**

***Index Terms*— Alpha shape, concept drift, nonstationary environment, semisupervised learning (SSL), verification latency.**

I. Introduction

THE fundamental assumption made by most of the learning algorithms generating a computational model is that the data are drawn from a fixed but unknown distribution. This assumption implies that future field (or test) data on which the model will be used come from the same distribution as the training data on which the model was developed in the first place. In many real-world applications, the fixed distribution assumption simply does not hold, rendering vast majority of traditional machine learning algorithms ineffective. As a result, new approaches, generally known as domain adaptation approaches, have been developed specifically to address the scenarios where training and test data come from different distributions, called source and target domains, respectively. Domain adaptation approaches, however, make other assumptions, such as having abundant labeled data from source domain, the supports of source and target domain being the same, and most importantly, limiting the problem to only one source and one target domain. Therefore, domain adaptation approaches are not suited for the so-called concept drift problem, created by nonstationary environments that continuously generate (streaming) data whose distributions may change over time.

Such nonstationary environments are increasingly common in real-world applications: network intrusion, web usage and user interest analysis, natural language processing, speech and speaker identification, spam detection, anomaly detection, analysis of financial, climate, medical, energy demand, and pricing data, as well as the analysis of signals from autonomous robots and devices are just a few examples. On the other hand, the vast majority of concept drift research on learning from nonstationary environments—including our prior efforts—have focused on supervised approaches [1]–[12]. The heavy reliance of supervised approaches on labeled data creates a significant limitation as streaming data are usually unlabeled and unstructured. This limitation is particularly acute if obtaining labeled data require expert annotation, which can be a very costly and time-consuming process. Algorithms that can learn using fewer labeled instances have therefore gained attention in related fields of semisupervised learning (SSL) and active learning (AL). In a nonstationary environment setting, SSL and AL approaches typically assume that, in addition to unlabeled data, some limited amount of labeled data are either readily available (for SSL algorithms), or may be requested (for AL algorithms), at each time step a new batch of data is received. Such regularly provided labeled data then allow classifiers to track the changes in class priors, class distributions, posterior distributions of class membership, or the number of target classes.

More recent research, typically referenced as verification latency, however, adds an important constraint: labeled data are not available at every time step, nor even in regular intervals, which significantly complicates the learning process. Verification latency, as denoted in [13], describes a scenario where true class labels are not made available until sometime after the classifier has made a prediction on the current state of the environment. The duration of this lag may not be known *a priori*, and may vary with time; yet, classifiers must propagate information forward until the model can be verified.

The motivation of this paper is to explore the extreme verification latency case, where the lag duration is set to

Manuscript received January 9, 2013; revised July 25, 2013; accepted July 25, 2013. Date of publication September 19, 2013; date of current version December 13, 2013. This work was supported by the National Science Foundation under Grant ECCS-0926159 and ECCS-1310496.

The authors are with the Signal Processing and Pattern Recognition Laboratory, Electrical and Computer Engineering Department, Rowan University, Glassboro, NJ 08028 USA (e-mail: dyerk@rowan.edu; robcapo@gmail.com; polikar@rowan.edu).

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

Digital Object Identifier 10.1109/TNNLS.2013.2277712

infinity—meaning no labeled data are ever received after initialization. We refer to this scenario as initially labeled streaming environment (ILSE), and propose a framework for learning in such an environment. This scenario removes many of the limitations and restrictions made in domain adaptation, SSL or AL. A theoretically justified solution to this extreme learning environment can then provide effective algorithms when labeled data are not available for extended periods, whether that period is finite or otherwise. Real-world examples of such an extreme learning setting are perhaps only a few today, but are rapidly growing because of massive automated and autonomous acquisition of sensor, web user, weather, financial transaction, energy usage, and other data. Furthermore, such applications can be extremely important: network intrusion with malicious software (malware) attacks—where malware programmers are able to modify the malware faster than network security can identify and neutralize it, is a major current day challenge. Creating a labeled database for this scenario is difficult and expensive, because the data—which arrive continuously (i.e., streaming)—need to be isolated on a virtual machine, features need to be extracted from the header data, and then evaluated by a human expert. Many automation applications provide other examples, such as robots, drones, and autonomous vehicles encountering surrounding environment changing at a pace too quick for a human to verify all the actions.

In this paper, we introduce the COMPacted Object Sample Extraction (COMPOSE) framework to address the extreme verification latency in an ILSE setting: learn drifting concepts from a streaming nonstationary environment that provides only unlabeled data after initialization. COMPOSE follows three steps to do so: 1) combine initial labels with new unlabeled data to train an SSL classifier and label the current unlabeled data; 2) for each class, construct α shapes (a generalization of convex hull), providing a tight envelope around the data that represent the current class conditional distribution; and 3) compact (i.e., shrink) the α shape and extract instances—called core supports—from the compacted α shape, which now represents the geometric center (core support region) of each class distribution. The process is repeated iteratively as new unlabeled data arrive, where the core supports from the previous iteration serve as the labeled instances for the current iteration.

We compare the ability of COMPOSE in tracking such an environment to that of: 1) Bayes classifier that is continuously updated with full access to the true distribution; 2) an ensemble-based nonstationary learning algorithm, Learn^{++}.NSE that is also trained in a fully supervised manner; and 3) Krempl's arbitrary subpopulation tracker (APT) algorithm that is recently proposed to address a similar extreme verification latency scenario.

In the following section, we provide an overview of domain adaptation, SSL, and AL algorithms followed by recent efforts using SSL and AL algorithms in nonstationary learning, specifically with regards to verification latency. A detailed description of the APT algorithm is also included. In Section III, we introduce and discuss the COMPOSE algorithm in detail, including the computational complexity of each of its components. In Section IV, we describe the preliminary experiments designed to determine the behavior of this proof-of-concept algorithm, and obtain a general understanding of its capabilities and limitations in learning an initially labeled nonstationary and streaming environment. We draw our conclusions, and discuss the strengths and current weaknesses of the algorithm, along with our future work in Section V.

II. Literature Survey

A. Domain Adaptation

The goal of domain adaptation approaches is to use ample labeled data obtained from one domain (source) and ample unlabeled data obtained from another domain (target) to develop a computational model that can predict well on the target domain, whose underlying distribution differs from that of the source domain. These approaches appear under various titles, such as domain adaptation [14], [15], inductive and transductive transfer learning [16], covariate shift [17], or multitask learning [18], all of which are related to sample selection bias, a concept that has been well known in the statistics community for some time [19]. In domain adaptation or transductive transfer learning, for example, it is assumed that the source and target domain joint data distributions are different (but related), i.e., $p_S(\boldsymbol{x},y) \neq p_T(\boldsymbol{x},y)$, where $\boldsymbol{x}$ and y are the instance and class labels, respectively. In inductive transfer learning, the source and target domain distributions are assumed the same, but the learning tasks represented in these domains are different (but related). In covariate shift, the most common form of domain adaptation, $p_S(y|\boldsymbol{x}) = p_T(y|\boldsymbol{x})$ with $p_S(\boldsymbol{x}) \neq p_T(\boldsymbol{x})$ is assumed, which may result—as in sample selection bias—when the training data may have been sampled more heavily from some regions of the feature space. Such cases are usually handled by instance weighting approaches, where likelihood function maximized by the learning algorithm is first weighted by a term proportional to $p_T(\boldsymbol{x})/p_S(\boldsymbol{x})$, called *importance* [17], [20], [21]. Another family of approaches is the so-called change of representation or feature representation approaches, where a transformation is made to source domain data for making its distribution similar to that of the target domain [15], [22]. Several assumptions have to be made, however, for any of these approaches to work. First, it is assumed that there are ample labeled data from the source domain, whereas the target domain provides ample unlabeled data, but little or no labeled data. Second, while the source and target distributions are different, it is assumed that they are not entirely independent, but are related. For example, for covariate shift, in addition to $p_S(y|\boldsymbol{x}) = p_T(y|\boldsymbol{x})$, it is also assumed that the support of $p_T(\boldsymbol{x})$, i.e., $\{\boldsymbol{x} : p_T(X = \boldsymbol{x}) > 0\}$ is contained in the support of $p_S(\boldsymbol{x})$. A thorough theoretical analysis of exactly how far the two distributions can be from each other is studied in [23] in terms of a new divergence metric. Once the conditions on how source and target distributions may differ from each other are met, domain adaptation approaches can often provide theoretical guarantees and bounds on performance, computational complexity, or the number of labeled instances required. A good

review of the domain adaptation approaches can be found in [7] and [24].

B. Semisupervised Learning

SSL uses limited labeled data to transfer their class information to unlabeled data following one or more of four general assumptions [25], [26]: 1) the smoothness or local consistency assumption: nearby instances must belong to the same class; 2) the cluster or global consistency assumption: instances in the same cluster must belong to the same class; 3) the low-density separation assumption: decision boundaries must lie in low-density regions; and 4) the manifold assumption: high-dimensional data reside on a lower dimensional manifold. All SSL algorithms use some variation of a common iterative recipe: train a classifier from available labeled data, classify the remaining unlabeled data, add instances whose confidence exceeds a threshold to the permanently labeled training set, and remove instances that did not meet this threshold. Note that SSL can be considered as a special case of domain adaptation, where labeled and unlabeled data represent source and target domains, respectively, with the difference in the underlying distributions limited to the noise in both data sets.

Several SSL algorithms are now well established, though primarily for use in static environments, and they typically fall into one of three general categories: 1) generative algorithms, such as [27] and [28], which assume that the data are provided by a fixed yet unknown distribution, and that the decision boundaries can be represented based on class posteriors; 2) low-density separation algorithms, such as [29] and [30], which use density information from unlabeled instances to modify a decision boundary created using only labeled data; and 3) graph-based algorithms, such as [31] and [32], which construct a graph, $G = (V, E)$ with vertices, V, representing instances and edges, E, representing relationships between vertices. Class information is transferred from labeled instances to neighboring unlabeled instances based on the relationship defined by the connecting edges. Some SSL algorithms have recently been modified or included in a wrapper-based approach enabling them to work in a nonstationary environment; these approaches are discussed in Section II-D.

C. Active Learning

The goal of AL [33]–[36] is to reduce the training cost by selecting key instances, whose labels—if available—would provide the most benefit for learning the underlying distribution. In stationary environments, such instances usually lie along the class boundaries; however, as demonstrated in this effort, instances selected from the core region of a distribution may be more advantageous in a nonstationary environment. The most restrictive limitation of AL, however, is the implicit requirement that the label must always be provided—at some cost—for any instance requested by the algorithm; this is not always a realistic assumption. In an ILSE, no labeled data are available—nor can be requested—after initialization, so AL approaches cannot be readily used in this setting. Therefore, COMPOSE is not an AL algorithm, though it does follow a conceptually similar approach of selecting those key instances (labeled by the SSL learner) to retain and merge with new incoming unlabeled data to be used in the next iterative step.

D. Concept Drift and Nonstationary Environments

Unlike domain adaptation problems where the goal is to learn from one domain to predict on another, concept drift algorithms deal with continuously drifting data distributions. In addition, the strict limitations on how $p_S(\boldsymbol{x})$ and $p_T(\boldsymbol{x})$, or $p_S(y|\boldsymbol{x})$ and $p_T(y|\boldsymbol{x})$ must be related to each other may not always be realistic in continuously drifting environments. Learning concept drift in a nonstationary environment is a challenging problem precisely because the rate and nature of the drift are not known *a priori*. The changes in the underlying distributions can be gradual or abrupt, cyclical or otherwise, though are often assumed to be limited in nature, where the changes follow some structures. Completely random fluctuations, of course, cannot be learned [21], [23], [37]. Lack of a standardized metric, however, leaves the definition of limited drift up to each researcher [11], [17], [21], [23], [38].

Adapting SSL algorithms to nonstationary environments is possible, but has received relatively little attention, and therefore is an area open for exploration. Of the recent SSL work that focus on nonstationary learning, most of the studies embrace the aforementioned assumption that some labeled data from the drifting distribution are available at every time step. As a result, much of the work in this field borrow ideas from supervised online learning paired with an SSL algorithm. Goldberg *et al*. [39], for example, combine online convex programming with SSL regularization (with a focus on manifold regularization), providing a framework for other SSL methods, including regularization of multiview learning and low-density separation techniques. They also add an optional AL component to select which instances to label [34]. Li *et al*. [40] combine a decision tree that grows as new data arrive with a clustering approach, where deviations between the clusters are used to identify a new or reoccurring concept at the tree leaves. In [41], we evaluated Gaussian mixture models (GMMs) trained on both labeled and unlabeled data, and matched the components of each GMM to determine labels in a transductive learning setting. The Masud *et al*. [42] approach creates microclusters for each batch of data by combining a cluster impurity-based SSL with the expectation–maximization algorithm. In their work, instances from new data are classified by choosing the class with the highest cumulative normalized frequency from the k-nearest neighboring microclusters in the ensemble. Zhang *et al*. [43], on the other hand, identify four types of streaming data: 1) labeled; 2) unlabeled data from the *same distribution* as the previous batch (where instances that arrive early in the new batch are assumed to be from the *same distribution* as those that arrived last in the previous batch); 3) labeled; and 4) unlabeled data from a similar but a similar but *drifted distribution*. A relational k-means and semisupervised support vector machine (S^3VM) are paired to classify unlabeled instances from the stream. Another recent work—though not limited to SSL—is proposed in [44], which uses conceptual representation models to map concepts of

the current distribution to a previously experienced concept for accommodating reoccurring concepts. Classifiers retained from previous steps (that experienced the mapped concept) are used for aiding classification.

E. Nonstationary Environments with Verification Latency

Addressing verification latency (where labeled data are not immediately available at every time step) in a nonstationary environment requires a different framework to propagate class information forward through several time steps of solely unlabeled data. Zhang *et al.* [45] have proposed combining ensemble of classifiers and clusters, an approach that works when labeled data are available at least intermittently. Batches that contain labeled data are used to create a classifier, whereas purely unlabeled batches are used to form clusters. The ensemble classifies new instances by a majority vote that includes label mapping between the classifiers and clusters of the ensemble.

More recent approaches have been proposed for data that can be represented as mixtures of particular parametric distributions. In these approaches, each class is represented as a mixture of subpopulations, and distributions of the unlabeled data from subpopulations are tracked and matched to those known subpopulations based on initial labeled data. Special cases where subpopulations are mixtures of fixed number of Gaussians is described in [46], and for cases, when drift is only due to the changes in priors of the subpopulations is addressed in [47] and [48]. Another example of this group of algorithms is Krempl's APT algorithm [49]. The APT algorithm makes the following assumptions: 1) the drift must be gradual and systematic that can be represented as a piecewise linear function; 2) each subpopulation to be tracked must be present at initialization, where a subpopulation is defined as a mode in the class conditional distribution (i.e., a bimodal class distribution would consist of two separate subpopulations to be tracked for that class); 3) the covariance of each subpopulation remains constant; and 4) the rate of drift remains constant. Observing these assumptions, incoming data are classified through a two-step procedure: 1) use expectation maximization to determine the optimal one-to-one assignment between the unlabeled data and the drift-adjusted labeled data, and 2) update the classifier to reflect the population parameters of newly received data and the drift parameters relating the previous time step to the current one. Establishing a one-to-one relationship while identifying drift requires an impractical assumption that the number of instances remains constant throughout all the time steps. Krempl rectifies this by establishing a relationship in a batch method—matching a random subset of the exemplars to random subset of new instances, where the new instances are sampled without replacement. Krempl suggests a bootstrap method that can make the one-to-one assignments more robust but at additional computational cost. In addition, because of assumption (b), which is common to all subpopulation-based approaches, APT cannot track a scenario that introduces a new class, or a new subpopulation, even when an existing population splits into subpopulations.

In [50], we introduced the original COMPOSE framework COMpacted POlytope Sample Extraction (COMPOSE) and demonstrated its ability to learn in a 2-D binary class ILSE, using a set of graphical methods that is well known in computer graphics, visualization, and animation, but surprisingly and largely missing from machine learning literature. It became clear, however, that some of the steps in the original version of this algorithm cannot be readily extended to higher dimensional and multiclass data. In this paper, we present the revised COMPOSE algorithm, which is computationally more efficient than its predecessor, but also is capable of handling higher dimensional and multiclass data. Most importantly, other than the aforementioned limited drift assumption made by all algorithms, COMPOSE removes other assumptions regarding the distributions, nature of drift, and the availability of labeled data. We should also emphasize that while COMPOSE is designed for the extreme ILSE scenarios (i.e., verification latency of infinity), COMPOSE is quite versatile and can readily and naturally accommodate environments that present labeled data at regular or intermittent intervals. The additional labeled data—when available—can be used for model verification, increasing robustness, and most importantly, greatly relaxing the algorithm's sole limited drift assumption, as discussed in the following.

III. COMPOSE

A. Central Premise of the Algorithm

COMPOSE is intended for nonstationary environments that face incremental or gradual (limited) drift, rather than abrupt drift. Gradual drift is often considered more challenging to detect than abrupt change, as the data distribution $p^t(\boldsymbol{x})$ at time t and $p^{t+1}(\boldsymbol{x})$ at time $t+1$ may have significant overlap, which makes distinguishing (detecting change between) the two difficult. COMPOSE turns this difficulty into an opportunity, and takes advantage for the overlapping nature of incrementally changing distributions at consecutive time steps. The entire COMPOSE process is presented in a block diagram with accompanying illustrations in Fig. 1. At $t = 0$, COMPOSE is provided with (possibly very few) labeled data, depicted by opposing classes of (red) squares and (blue) circles [Fig. 1(a)], and relatively abundant unlabeled data, represented by (black) diamonds [Fig. 1(b)]. At all other time steps t, COMPOSE receives only unlabeled data. An SSL algorithm is trained with the labeled and unlabeled data, to label the currently unlabeled instances, as indicated with the change of color and shape in Fig. 1(c). COMPOSE creates an α-shape boundary object from the current data, defining a tight envelope representing the distribution of each class. Class boundaries are represented by solid outlines, enveloping shaded regions in Fig. 1(d). The boundary object for each class is compacted (i.e., shrunk) by a specified percentage, the compaction percentage (CP), to determine the core support region of each distribution, as shown by the darker shaded region with dashed outline in Fig. 1(e). Instances drawn from the core support region of the current distribution $p^t(\boldsymbol{x})$ are the most likely candidates to represent data drawn from the next distribution $p^{t+1}(\boldsymbol{x})$ that may have experienced translational, rotational, or volumetric

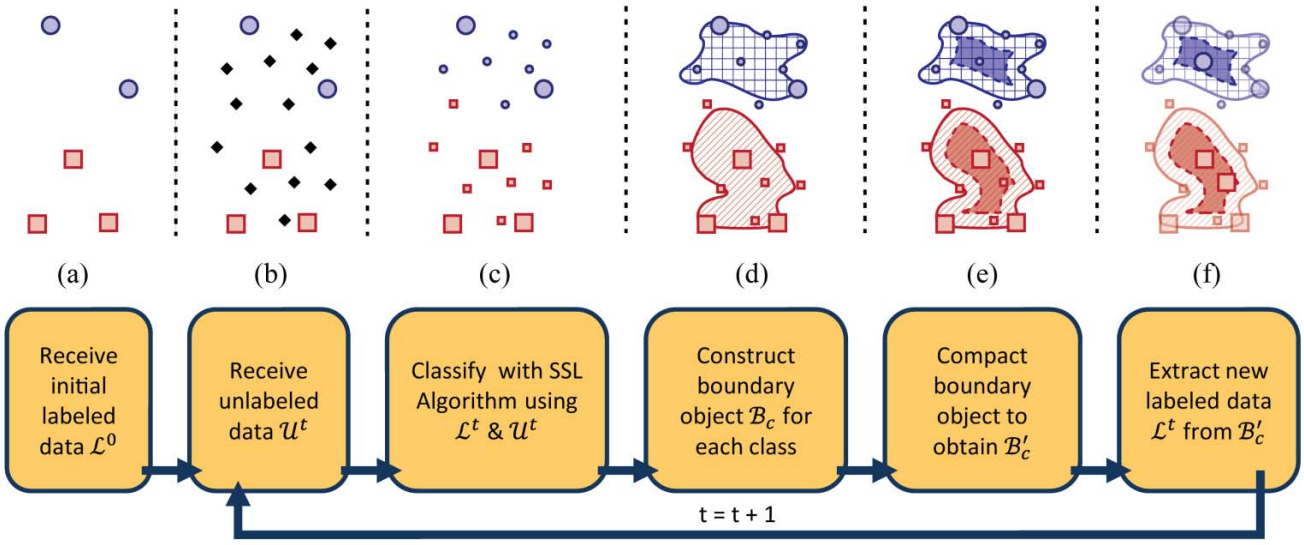

Fig. 1. Block diagram and graphical representations of corresponding stages of COMPOSE. (a) Receive initial labeled data. (b) Receive new unlabeled data (black diamonds). (c) Classify unlabeled data using SSL algorithm. (d) Construct alpha shape (boundary object, represented as shaded region enveloping the data) for each class. (e) Shrink the boundary object to obtain *core support region* for each class (dark shaded regions). (f) Extract labeled data - *the core supports* - from the core support region.

(i.e., expansion/contraction) drift. The final step of one iteration of COMPOSE extracts (now labeled) instances from the core support region(s) to be used as labeled data in the near future—these instances are referred to as core supports of that class [Fig. 1(f)]. It is possible to have multiple core support regions for any class. When new unlabeled data are received, they are combined with the core supports to retrain an SSL algorithm to adapt the drifting (nonstationary) environment, as COMPOSE iteratively updates itself. The progression of a single class over a series of time steps is shown in Fig. 2, experiencing (a) translation, (b) rotation, and (c) compaction. In each case, the core region from the previous time step (whose boundaries are shown with dashed lines) provides labeled instances for the current time step. It is important to emphasize that—unlike traditional SSL or AL algorithms used in nonstationary settings—all future labeled data are earned (generated) by COMPOSE (through core support extraction), and not paid for, purchased or requested from the user.

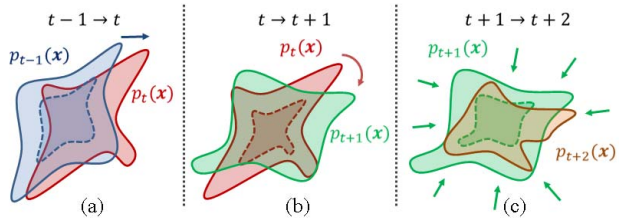

Fig. 2. Progression of a single class experiencing (a) translational, (b) rotational, and (c) volumetric drift.

B. Algorithm Description

Conventional SSL algorithms used in stationary environments require sufficient amount of labeled as well as unlabeled data. In a nonstationary ILSE, not only future labeled data are rare or nonexistent, data also drift, preventing conventional SSL algorithms from learning in such a setting. COMPOSE is designed to address this limitation by extracting relevant data, labeled by the SSL learner in the current time step, to be combined with the next batch of unlabeled data. This important modification allows SSL algorithms to be used in nonstationary environments.

The distribution $p^t(\boldsymbol{x})$ providing the unlabeled data at time t may have drifted from the distribution $p^{t-1}(\boldsymbol{x})$ at time $t-1$. Consistent with other nonstationary environment algorithms, we assume limited (gradual) drift, such that the extracted labeled data overlap the newly received unlabeled data. Therefore, the distribution $p^t(\boldsymbol{x})$ must overlap with the distribution $p^{t-1}(\boldsymbol{x})$, as shown in Fig. 2. This requirement is much less restrictive than the $p_S(y|\boldsymbol{x}) = p_T(y|\boldsymbol{x})$ requirement in covariate shift, or the support of $p_T(\boldsymbol{x})$ must be contained in the support of $p_S(\boldsymbol{x})$ requirement in general domain adaptation. As the amount of overlap between distributions of subsequent time steps increase, so does the ability and performance of COMPOSE in tracking the nonstationary distribution. The remainder of this section explains in detail how COMPOSE: 1) creates α shapes from the data; 2) compacts (shrinks) the α-shapes to create core regions; and 3) extracts core supports from the compacted α shapes to serve as labeled data for future time steps. The outline of the algorithm is listed in Fig. 3.

The algorithm has three inputs: 1) BaseClassifier, which can be any SSL algorithm, for classifying unlabeled data at each time step, t; 2) α, specifying the level of detail of the α-shape boundary object; and 3) CP. The algorithm is initialized at $t = 0$ with a set of labeled data, $\mathcal{L}^0 = \{\mathbf{x}_l^t \in X\}$, and corresponding labels, $\mathcal{Y}^0 = \{y_l^t \in Y = \{1, \ldots, C\}\}$, $l = 1, \ldots, M$ where M is the total number of labeled instances, and C is the total number of classes (step 1 in Fig. 3). At each subsequent time step t, new unlabeled data $\mathcal{U}^t = \{x_u^t \in X\}$ are received, $u = 1, \ldots N$ where N is the

Inputs: SSL algorithm – **BaseClassifier**; α-shape detail level – α; compaction percentage - CP
1. Receive labeled data
$\mathcal{L}^0 = \{\boldsymbol{x}_l^t \in X\}, \mathcal{Y}^0 = \{y_l^t \in Y = \{1, \ldots, C\}, l = 1, \ldots, M\}$
Do for $t = 0,1, \ldots$
2. Receive unlabeled data, $\mathcal{U}^t = \{x_u^t \in X, u = 1, \ldots, N\}$
3. Call **BaseClassifer** with $\mathcal{L}^t$, $\mathcal{Y}^t$, and $\mathcal{U}^t$
Obtain $h^t: X \rightarrow Y$,
Let $\mathfrak{D}^t = \{(\boldsymbol{x}_l^t, y_l^t): x \in \mathcal{L}^t \forall l\} \cup \{(\boldsymbol{x}_u^t, h_u^t): x \in \mathcal{U}^t \forall u\}$
4. Set $\mathcal{L}^{t+1} = \emptyset$, $\mathcal{Y}^{t+1} = \emptyset$
Do for each class $c = 1, \ldots, C$
5. Construct α-shape boundary, $\mathcal{B}_c = f(\alpha, \mathfrak{D}_c^t)$
Do Until number of core supports $|CS_c| = CP * |\mathfrak{D}_c^t|$
6. Compact α-shape boundary, $\mathcal{B}_c^{'} = g(\mathcal{B}_c)$
End
7. Extract core supports, $CS_c = \{x: x \in \mathcal{B}_c^{'}\} \cup \mathfrak{D}_c^t$, and add to labeled data for next time step
$\mathcal{L}^{t+1} = \mathcal{L}^{t+1} \cup CS_c$
$\mathcal{Y}^{t+1} = \mathcal{Y}^{t+1} \cup \{y_u: u \in [|CS_c|], y = c\}$
End
End

Fig. 3. COMPOSE psuedocode.

total number of unlabeled instances (step 2). Both labeled and unlabeled data are passed to BaseClassifier for generating a hypothesis $h^t : X \rightarrow Y$. A combined data set $\mathcal{D}^t$ is constructed by merging $\mathcal{L}^t$ and $\mathcal{U}^t$, where the class labels for $\mathcal{U}^t$ are provided by h^t (step 3). With labels for all the instances of $\mathcal{D}^t$ now available, COMPOSE then extracts core supports for each class, selected from the core support region of the current distribution (steps 4–7). The underlying premise here is that the core support region of the data at the current time step—compared with any other time step—is most likely to have maximum overlap with the drifted distribution in the next time step, regardless of the nature of drift. Therefore, these core supports can be used to serve as labeled data for the next time step's SSL classifier. Specifically, the labeled data set for the next time step $(\mathcal{L}^{t+1}, \mathcal{Y}^{t+1})$ is first initialized as an empty set (step 4). For each class, $c = 1, \ldots, C$ identified by h^t an α-shape class boundary object $\mathcal{B}_c$ is constructed using the method described in Section III-C (denoted as function $f(\blacksquare)$ in step 5). The class boundary object $\mathcal{B}_c$ is then compacted using the method described in Section III-D to produce the core support region $\mathcal{B}_c^{'}$ (denoted as function $g(\blacksquare)$ in step 6) such that desired core supports specified by CP are obtained. Then, all the instances that reside in the compacted region $\mathcal{B}_c^{'}$ are core supports and are retained to serve as labeled data for the next time step. Core supports obtained from each class are appended to finalize the labeled data $(\mathcal{L}^{t+1}, \mathcal{Y}^{t+1})$ in step 7.

C. α-Shape Construction Function

We first introduce the basic terminology used within the context of constructing α shapes. A d-simplex, or simply a simplex, is the convex hull of $d + 1$ vertices connected via edges, where d is the dimensionality of the data (e.g., a 2-simplex is a triangle defined by three vertices and a 3-simplex is a tetrahedron defined by four vertices). In graph-based methods, each vertex represents a data point, an instance with d features. Each d-simplex is constructed from multiple $(d - 1)$-simplexes, called faces (e.g., each face of a triangle is a line and each face of a tetrahedron is a triangle). The circumsphere of a simplex is the hypersphere uniquely defined by the vertices of a simplex (e.g., a circle is defined by the three vertices of the triangle it circumscribes; and a sphere is defined by the four vertices of the tetrahedron it circumscribes).

An α shape is a set of connected faces creating a hull that describes a finite set of points at a specified level of detail, defined by the free parameter $\alpha > 0$. For a sufficiently large α, the resultant α shape is the convex hull of the points. As α decreases, the α shape may become concave, form holes, or include completely disconnected regions. These three aspects of α shapes make them attractive for machine learning as they can properly represent voids and nested classes that many algorithms using convex hulls or other simpler methods (such as calculating the centroid of a distribution) cannot. Fig. 4 shows how α changes the representation of a data set in an α shape. Fig. 4(a) shows a large α resulting in the convex hull of the (blue) diamonds including a large region void of data, as well as an opposing class of (red) circles. As α decreases in Fig. 4(b)–(d), the true feature space from which the set of diamonds was sampled becomes more apparent—the letter P. If α is, however, chosen too small, as in Fig. 4(e), the α shape becomes a group of disconnected regions, which is undesirable. The α parameter can be chosen heuristically, based on prior knowledge or experience, or based on sample density as proposed in [51].

The pseudocode of the α-shape construction function is shown in Fig. 5, whose inputs are: 1) single-class data $\mathcal{D}$ (as labeled by the SSL in the previous step of the algorithm) and 2) the α parameter specifying the desired level of detail. α-shape construction begins with a Delaunay tessellation of $\mathcal{D}$ (step 1 in Fig. 5). Delaunay tessellations are an extension of Delaunay triangulations into higher dimensions. Delaunay tessellations nest simplexes such that no point in the set may lie inside the circumsphere of any simplex in the tessellation. The union of all the simplexes in the tessellation produces the convex hull of the set. There are several algorithms that accomplish Delaunay tessellations; we have used the Quickhull algorithm [52], denoted as $Q(\blacksquare)$ in step 1, for its speed and relative lower complexity whose upper bound is $\mathcal{O}(n^{\lfloor (d+1)/2 \rfloor})$, where n is the number of points in the set, d is the dimensionality, and $\lfloor \blacksquare \rfloor$ is the floor function.

The α-shape $\mathcal{B}$ is initially set to be equal to convex hull defined by the Delaunay tessellation (step 2). Each face $\mathcal{F}$ is subsequently analyzed, categorized and, if necessary, certain simplexes containing that face are removed to produce the final α shape (steps 3–5). To do so, we first iterate through every face, and identify the two simplexes, s_1 and s_2, that share $\mathcal{F}$ (step 3 and Fig. 6). The radii of the circumspheres of each simplex are then calculated by passing the simplex's vertices to the circumsphere radius function [denoted $r(\blacksquare)$ in step 4, and described in the following]—the smaller radius is labeled μ_1 and the larger as μ_2 (Fig. 6). If $\mathcal{F}$ is located at the edge of

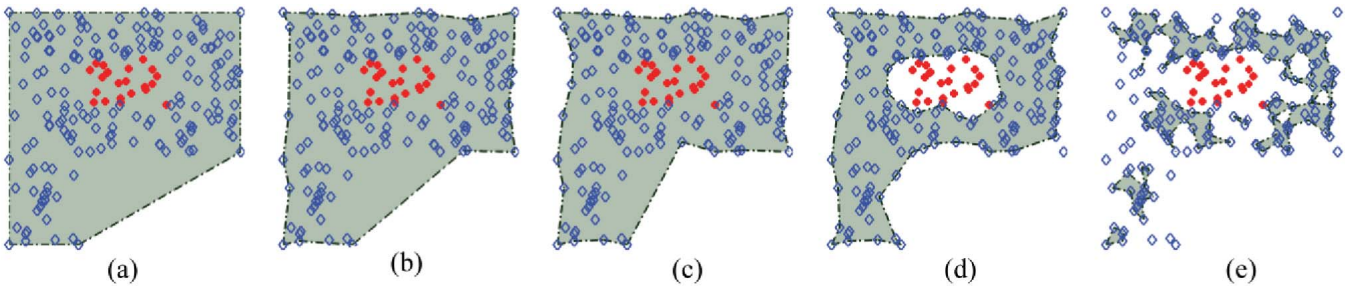

Fig. 4. Shaded region demonstrates α-shape on set of blue diamonds at different levels of detail specified by α. (a) $\alpha = \infty$. (b) $\alpha = 0.5$. (c) $\alpha = 0.25$. (d) $\alpha = 0.1$. (e) $\alpha = 0.05$.

Input: α-shape probing radius – α; Data features – $\mathfrak{D}$
1. Construct Delaunay tessellation of data, $T = Q(\mathfrak{D})$
2. Initialize α-shape as Delaunay tessellation $\mathcal{B} = T$
Do for each face, $\mathcal{F} \in T$
3. Find simplexes, s_1 and $s_2 \in T$, that share $\mathcal{F}$
4. Find radii of circumspheres, $\mu = r(\blacksquare)$
If $\mathcal{F}$ is an edge of T
Radius of simplex, $\mu_1 = r(s_1)$
Denote as boundary, $\mu_2 = Inf$
Else
$\mu_1 = \min[r(s_1), r(s_2)]$
$\mu_2 = \max[r(s_1), r(s_2)]$
End If
5. Categorize $\mathcal{F}$ and update $\mathcal{B}$ accordingly
Case 1: $\alpha > \mu_2$ $\mathcal{F}$ is interior
Case 2: $\mu_1 < \alpha < \mu_2$ $\mathcal{F}$ is regular, $\mathcal{B} = \mathcal{B} \backslash \{s_2\}$
Case 3: $\alpha < \mu_1$ $\mathcal{F}$ is singular, $\mathcal{B} = \mathcal{B} \backslash \{s_1, s_2\}$
End

Fig. 5. α-shape compaction function.

the tessellation (i.e., it is not shared by a second simplex), the radius of the (nonexistent) second simplex is set to infinity, $\mu_2 = \infty$.

The simplex passed to the circumsphere radius function is defined by its $d + 1$ noncoplanar vertices (instances) $\mathbf{x}_p$, $p = 1, \ldots, d + 1$, each vertex defined by d coordinates (features)

$$\mathbf{x}_p = \left\{x_{p_1}, x_{p_2}, \ldots, x_{p_d}\right\}. \tag{1}$$

From the equation for circumcircle of a triangle [53], extended to higher dimensions, the equation of the circumsphere is as follows:

$$\begin{vmatrix} \sum_d x_{\blacksquare_d}^2 & x_{\blacksquare_1} & x_{\blacksquare_2} & \cdots & x_{\blacksquare_d} & 1 \\ \sum_d x_{1_d}^2 & x_{1_1} & x_{1_2} & \cdots & x_{1_d} & 1 \\ \sum_d x_{2_d}^2 & x_{2_1} & x_{2_2} & \cdots & x_{2_d} & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \sum_d x_{(d+1)_d}^2 & x_{(d+1)_1} & x_{(d+1)_2} & \cdots & x_{(d+1)_d} & 1 \end{vmatrix} = 0 \tag{2}$$

where $\mathbf{x}_\blacksquare$ is used to represent any point (instance) on the hypersphere and $x_{\blacksquare_d}$ is its dth feature. Cofactor expansion of the first row, valid for any point residing on the hypersphere, produces the equation of a hypersphere in general form

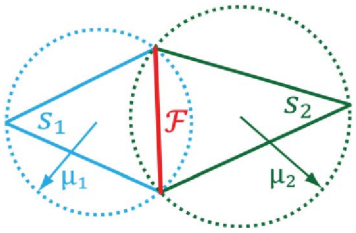

Fig. 6. Face (centered in red) to be classified is shared by simplex with smaller radius on left (blue) and simplex with larger radius on right (green).

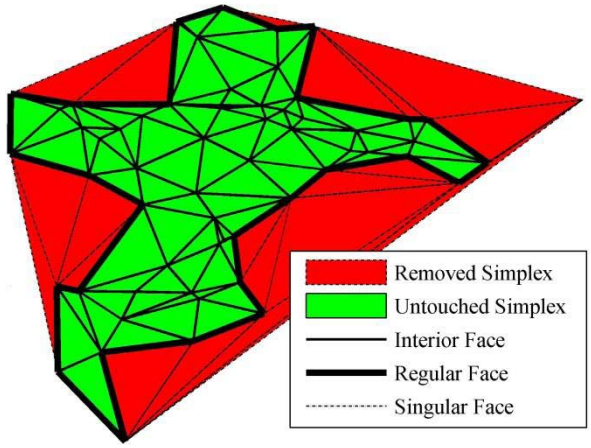

Fig. 7. Sample α shape showing simplexes in Delaunay tessellation and how faces are classified in relation placement in α shape.

$$\sum_d x_{\blacksquare_d}^2 \mathbf{M}_{11} + \left[\sum_d (-1)^d (x_{\blacksquare_d}) \mathbf{M}_{1(d+1)}\right] + \mathbf{M}_{1(d+2)} = 0 \tag{3}$$

where $\mathbf{M}_{ij}$ is a matrix minor—the determinant of the matrix after removing row i and column j. The result after completing the square and rearranging the terms is the standard form of a hypersphere

$$\sum_d \left(x_{\blacksquare_d} - x_{0_d}\right)^2 = r^2 \tag{4}$$

where

$$x_{0_q} = (-1)^{q+1} 0.5 \frac{M_{1(q+1)}}{M_{11}}, \quad q = 1, \ldots, d \tag{5}$$

$$r^2 = \sum_d x_{0_d} - \frac{M_{1(d+2)}}{M_{11}} \tag{6}$$

with $\mathbf{x}_0$ and r are the center and radius of the hypersphere, respectively.

Once computed, the radii of the simplexes are compared to α to determine if the face is interior, regular, or singular

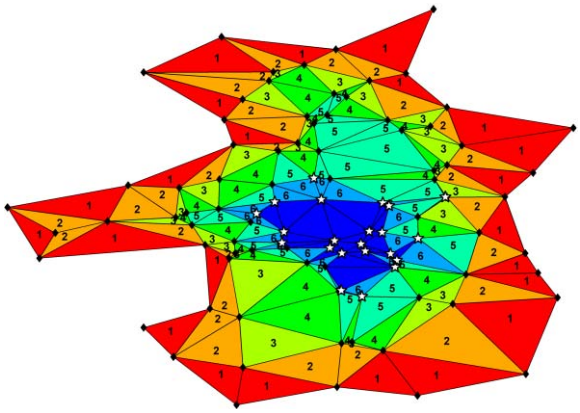

Fig. 8. Graphical representation of compaction method. Layers are removed in numerical order starting with (red) one and ending with (blue) six until cores supports remain, which are represented by (white) stars. CP used for this figure was 85%.

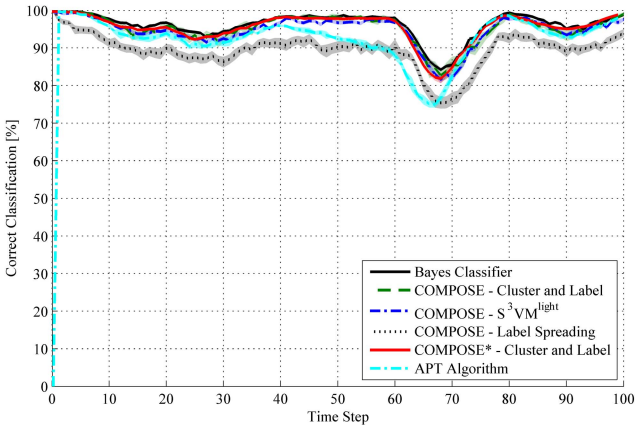

Fig. 9. Performance of updated COMPOSE (denoted by *) compared with the original COMPOSE on the unimodal Gaussian experiment. Parameters: $\alpha = 0.4$ and CP $= 0.70$. For a video illustrating the nonstationary environment for this experiment, see http://users.rowan.edu/~polikar/research/TNNLS/.

(step 5 in Fig. 5). An interior face, where $\alpha > \mu_2$, is completely encapsulated by the final α shape resulting in both simplexes that share this face to remain within the α shape. A regular face, where $\mu_1 < \alpha < \mu_2$, defines the boundary of the α shape (dark black faces in Fig. 7). As a result, the simplex with the larger radius circumsphere (shown by the red regions and exterior of dark black faces in Fig. 7) is removed from the α shape, while the simplex with the smaller radius circumsphere remains. A singular face, where $\alpha < \mu_1$, as described in [54], traditionally has two subcategories: 1) attached and 2) unattached. In either case, both simplexes are removed; however, the shared edge remains protruding from the α shape as a spoke in the attached subcategory. The use of α shapes in COMPOSE does not require differentiation between these two subcategories, as the singular attached case always disappears during the α-shape compaction function described in Section III-D. Hence, all singular faces and both simplexes that share the singular face are removed from the final α shape. Examples of each type of edge and the resultant α shape after simplexes have been removed are shown in Fig. 7. While an α shape is traditionally defined as the union of all regular faces, it suffices for COMPOSE to define an α shape to be the union of all simplexes not removed from the Delaunay tessellation, as described in Section III-D.

Timing tests confirmed that the computational complexity of this step of COMPOSE increases exponentially with dimensionality, and hence is the most expensive module of the algorithm. We discuss methods to reduce complexity in Section V.

D. α-Shape Compaction Function

Compaction of the α-shape boundary object is achieved by iteratively removing a layer of simplexes from the edges of the α shape, as if unwrapping an onion, until the desired CP is achieved. The compaction threshold is found by multiplying the number of instances in the initial α shape by (1-CP), yielding the target number of instances to remove. Each time a layer of simplexes is peeled off, the number of instances in the compacted α shape is reduced. Compaction is complete when the number of remaining core supports is fewer than the compaction threshold.

This method is illustrated in Fig. 8, where each removed layer of simplex is numbered in the order it is removed. The first (outermost) layer removed is stated by one and shaded in red; the last layer is in light blue and contains six. The data remaining after the compaction become the core supports, shown by white stars clustered at the center of the α shape.

Identifying which simplexes reside at the edge of an α shape is a simple task, as boundary simplexes have one or more faces that are not shared with another simplex. By creating a list of all faces and identifying to which simplex each belongs, a simple sort can identify unmatched faces. The simplex identifier associated with the unmatched faces are the simplexes located at the edge of the α shape. The complexity of this method is $\mathcal{O}(s^2)$, where s is the total number of simplexes in the α shape, which is linearly related to the total number of instances. This compaction function, unlike the original skeleton-based compaction algorithm mentioned in [50], is independent of dimensionality, and hence significantly reduces the complexity of the overall approach.

IV. Experiments and Results

A. Experimental Setup and Results on Synthetic Data Sets

We designed two new experiments and repeated two experiments from [50], using nonstationary Gaussian data, to demonstrate that the new COMPOSE framework: 1) performs just as well, if not better, than the initially proposed framework; 2) is able to extend to higher dimensions; and 3) can adapt to the introduction of a new class (see video links in figure captions). In addition to COMPOSE, we have repeated each experiment with the APT algorithm (the only other algorithm currently available for the extreme verification latency problem), and the optimal Bayes classifier to provide an upper bound performance. The results are shown in Figs. 9–12. The Bayes classifier was trained in a fully supervised manner, having full access to correct labels for all instances at all time steps. This is a scenario that is deliberately designed to be unfair against

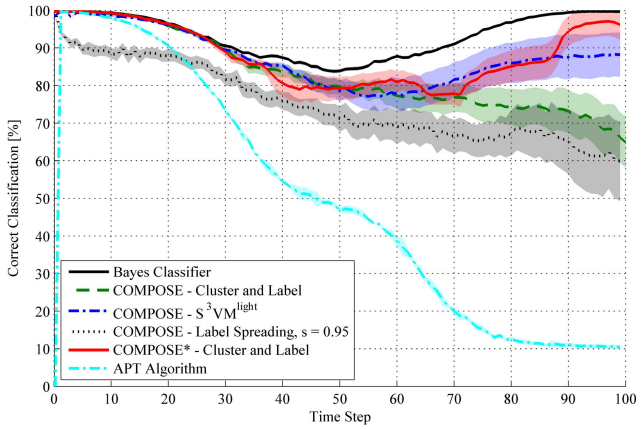

Fig. 10. Performance of updated COMPOSE (denoted by *) compared with the original COMPOSE on the multimodal Gaussian experiment. Parameters: $\alpha = 0.43$ and CP $= 0.70$. For a video illustrating the nonstationary environment for this experiment, see http://users.rowan.edu/~polikar/research/TNNLS/.

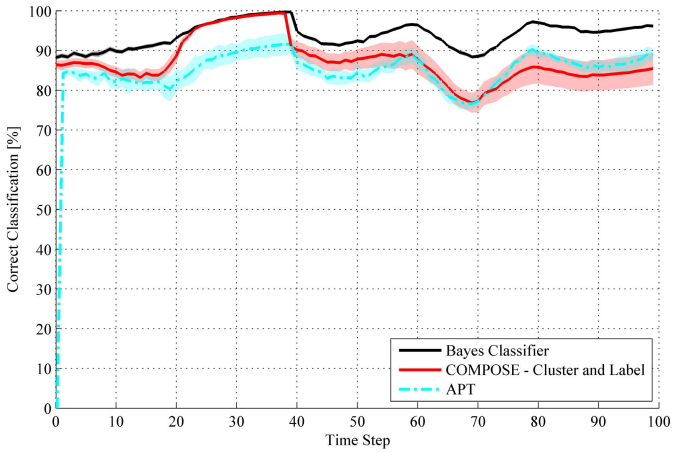

Fig. 11. Performance of COMPOSE in 2-D unimodal Gaussian experiment where a third class is added at time step 40. Parameters: $\alpha = 0.6$ and CP $= 0.60$. For a video illustrating the nonstationary environment for this experiment, see http://users.rowan.edu/~polikar/research/TNNLS/.

COMPOSE and APT, as these algorithms maintained the ILSE assumption where labels were provided only for a subset of the data and only during the initial time step. All comparisons with Bayes classifier should be interpreted within this context.

In this group of experiments, we used Gaussian distributions starting at some initial state at some arbitrary time $t = 0$. COMPOSE was initialized using only 5% of randomly selected data labeled, though we ensured each class is represented by at least one labeled instance; APT, however, was initialized with a full set of labeled data. At each subsequent step t, the distributions drift according to some parametric equations (shown in Tables I and II, and Figs. 13 and 14 for the new experiments), with 100 new unlabeled instances presented per Gaussian distribution. The experiments end after 100 steps, at some arbitrary time, $t = 1$.

All experiments were repeated 50 times for COMPOSE and five times for APT, providing the 95% confidence intervals shown with the shaded regions around the performance curves. APT was run only five times because of its significantly longer computation time as discussed in the following section. APT

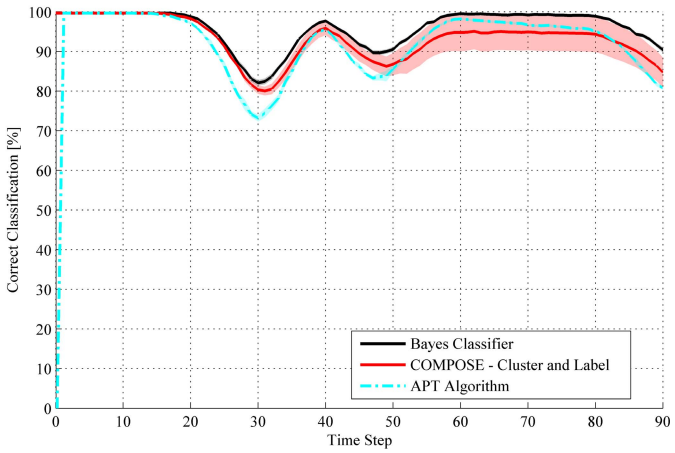

Fig. 12. Performance of COMPOSE in 3-D unimodal Gaussian experiment. Parameters: $\alpha = 3.1$ and CP $= 0.60$. For a video illustrating the nonstationary environment for this experiment, see http://users.rowan.edu/~polikar/research/TNNLS/.

was run with a maximum of five expectation–maximization iterations per time step and no initial drift information provided at initialization (providing drift rates known *a priori* is an option of APT) to keep the comparison with COMPOSE fair. COMPOSE's independence of SSL algorithm used as the BaseClasifier was demonstrated in [50], therefore, new experiments in this paper are presented with cluster-and-label chosen as the SSL algorithm. This algorithm was selected because of minimal free parameters it needs, and its ability to easily adapt a multiclass problem—unlike, e.g., S^3VM, which does not readily work in multiclass problems.

There are several variations of cluster-and-label; we used k-means to perform the clustering, and majority vote of labeled instance in the clusters for labeling the clusters. The algorithm begins with $k = 5$, the number of clusters to find, which iteratively reduces itself by one if it is unable to find a solution where every cluster contains at least one labeled point. COMPOSE free parameters (α and CP) were selected heuristically (shown within figures), were not optimized, and remained fixed throughout the experiments.

1) Repeated Unimodal and Multimodal Gaussians: The two experiments repeated from [50] serve as a benchmark, comparing the original COMPOSE (skeleton-based compaction, limited to 2-D) to the updated framework presented here. The experiments were governed by parametric equations, similar to those of Tables I and II for new experiments shown below. As shown in Figs. 9 and 10, the modified COMPOSE framework (denoted by solid red line) performs better in both experiments when compared with its earlier counterpart (using cluster-and-label as the SSL). Performance of the original COMPOSE algorithm with other SSL algorithms is also shown for comparison.

During the periods of increased class overlap, time steps 60–70 in Fig. 9, COMPOSE outperforms APT with statistical significance. During the other steps, both algorithms have similar performances, closely tracking Bayes classifier (black curve).

The primary weakness of APT—the assumption that all subpopulations must be present at initialization—is most

TABLE I

PARAMETRIC EQUATIONS GOVERNING DRIFT OF (NEW) 2-D GAUSSIAN EXPERIMENT WITH ADDED CLASS

Class	$0 \le t < 0.2$				$0.2 \le t < 0.4$				$0.4 \le t < 0.6$			
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y
C1	2 - 5t	5	1.5	5 - 5t	1	5 - 10t	1.5 + 7.5t	3	1	3 - 5*t	3 - 10t	3 - 10t
C2	5 - 5t	8	5 - 15t	1.5 + 2.5t	4 + 20t	8	2	2	8	8 - 20t	2 - 5t	2 + 10t
C3	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	5	5 + 15t	1 + 5t	1 + 5t

Class	$0.6 \le t < 0.8$				$0.8 \le t \le 1$			
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y
C1	1 - 5t	2 + 15t	1 + 15t	1	0 + 5t	5 + 15t	4 - 10t	1 + 10t
C2	8	4 + 20 t	1	4 - 10t	8	8 - 30t	1 + 5t	2
C3	5 + 5t	8 - 30t	2	2 +5t	6 - 25t	2	2 + 5t	3

TABLE II

PARAMETRIC EQUATIONS GOVERNING DRIFT OF 3-D GAUSSAIN EXPERIMENT

Class	$0 \le t < 0.2$			$0.2 \le t < 0.4$			$0.4 \le t < 0.6$		
	μ_x	μ_y	μ_z	μ_x	μ_y	μ_z	μ_x	μ_y	μ_z
C_1	9 -25t	1 + 10t	8 - 15t	4 - 10t	3 + 15t	5 - 15t	2 + 15t	6 + 15t	2 - 5t
C_2	0 + 10t	0 + 10t	3 - 10t	2 + 20t	2 + 20t	1 + 10t	6 - 20t	6 + 10t	3 + 10t

Class	$0.6 \le t < 0.8$			$0.8 \le t \le 1$		
	μ_x	μ_y	μ_z	μ_x	μ_y	μ_z
C_1	5 + 25t	9 + 5t	1 - 5t	10 - 15t	10 - 10t	0 + 15t
C_2	2 + 25t	8	5 - 10t	7 - 10t	8 + 6t	3 - 5t

Covariance matrices remain fixed as an identity matrix, **I**, for each class throughout experiment

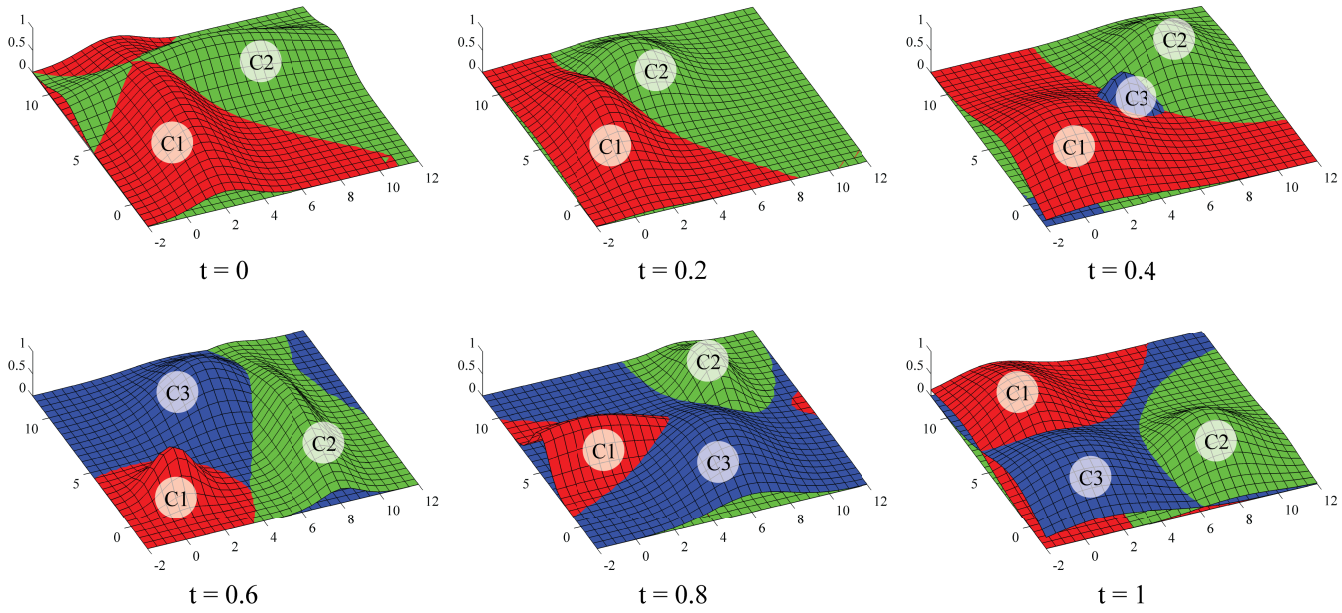

Fig. 13. Snapshots of Gaussian added-class experiment at intervals where parametric equations governing drift change. Note that the new (third) class appears at $t = 0.4$, and then spreads into the feature space. See video illustrating the experiment at http://users.rowan.edu/~polikar/research/tnnls.

vividly seen in the second experiment that featured a scenario that splits a unimodal distribution into a multimodal distribution, which have then merged to return to a unimodal distribution later (see link in Fig. 10 caption for visualization of this scenario). APT failed to track these diverging distributions, as shown in Fig. 10, because the diverging distribution created a new subpopulation that APT did not know at initialization. COMPOSE, however, was able to track the distributions before the split, throughout the split, as well as after their merge. Furthermore, COMPOSE was able to follow the performance of Bayes closely. This is a quite noteworthy, considering the unfair circumstances under which COMPOSE operates against the Bayes classifier.

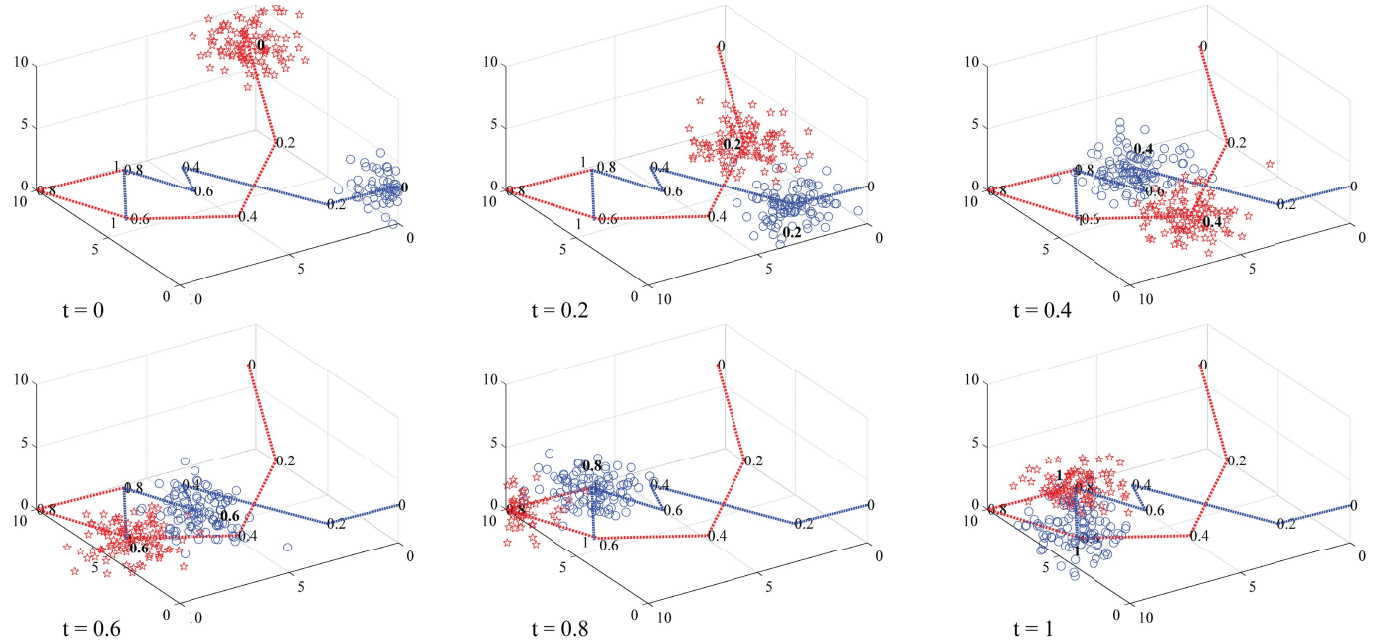

Fig. 14. Snapshots of 3-D data sampled from a Gaussian distribution shown at time steps where parametric equations governing drift change. See video illustrating the experiment at http://users.rowan.edu/~polikar/research/tnnls.

2) Unimodal Gaussian With Added Class: This new experiment initializes two Gaussian distributions at $t = 0$, and then adds a third class at time step 40, as governed by the parametric equations of Table I, and as shown in Fig. 13. The third class was added with only 5% of its data labeled—with labels provided only at this time step—which constitutes the initialization of the new class for COMPOSE. In contrast, the full training set (i.e., all instances labeled) for the new class was provided to APT. We also note that the labeled data provided at this time step came only from the new class to comply with ILSE requirements. Fig. 11 compares COMPOSE performance against that of APT and Bayes classifier. COMPOSE outperforms APT with statistical significance during certain time intervals (time steps 20–60). During other times, the differences in performances were not statistically significant. All classifiers experience a performance drop when the new class is added, which is expected.

3) Unimodal Gaussians in 3-D: This new experiment, governed by equations of Table II and illustrated in Fig. 14, extends the feature space to 3-D to demonstrate (and graphically illustrate) that revised COMPOSE can actually scale to higher dimensions (also see 8-D real-world data set below). Fig. 12 compares COMPOSE's generalization performance to that of Bayes classifier and APT. The important observation here is that COMPOSE can still follow Bayes extremely well, despite the unfair nature of the experimental setup, and outperforms APT with statistical significance during the more difficult periods of high overlap, and performing similarly during other time steps.

B. Experimental Setup and Results of Real-World Data

We have also tested COMPOSE using the National Oceanic and Atmospheric Administration (NOAA) weather data set collected over a 50-year span from Offutt Air Force Base in Bellevue, Nebraska. Eight features (temperature, dew point, sea-level pressure, visibility, average wind speed, max sustained wind speed, and minimum and maximum temperature) are used to determine whether each day experienced rain or no rain. The data set contains 18 154 daily readings of which 5693 are rain and the remaining 12 461 are no rain. Data were grouped into 49 batches of one-year intervals, containing 365 instances (days) each; the remaining data were placed into the fiftieth batch as a partial year.

This experiment was initialized with 5% of the 365 instances labeled. Every subsequent time step received the full set of additional 365—all unlabeled—instances. Since this is real-world data (and not drawn from a distribution), and since all the available data are presented at each time step, only one trial is possible. Repeating trials would result in the same performance each time so a confidence interval cannot be obtained. In [4], this data set was used to test an ensemble of supervised learners (Learn^{++}.NSE—for non stationary environments) receiving labeled data with every time step in a seasonal fashion—batches of 90 instances. We compare yearly batch performance of COMPOSE and APT with that of Learn^{++}.NSE (with SVM as well as naïve Bayes used as BaseClassifier) in Fig. 15. COMPOSE greatly outperforms APT, but the most compelling demonstration of COMPOSE's performance comes from comparing COMPOSE with Learn^{++}.NSE. COMPOSE trained in an ILSE setting (and with only 18 labeled instances), is quite competitive with an ensemble of classifiers that are trained in an entirely supervised manner, receiving fully labeled data at every time step.

C. Computation Time Tests

As the experiments have shown, COMPOSE can learn in an initially labeled streaming nonstationary environment, and

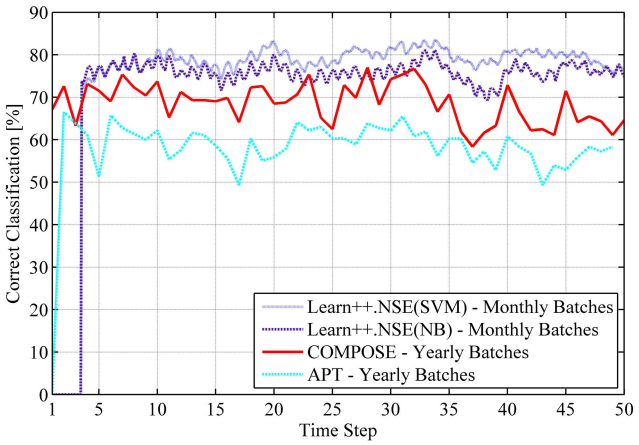

Fig. 15. Performance of COMPOSE on the NOAA weather data set plotted against Learn^{++}.NSE and the APT algorithm. COMPOSE parameters: $\alpha = 14$ and CP $= 0.65$.

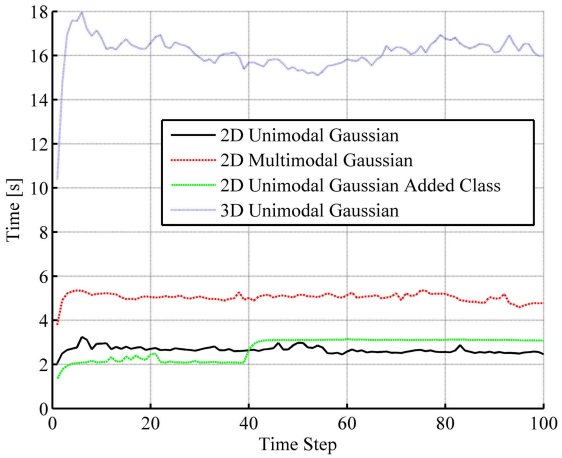

Fig. 16. Computation time of experiments.

successfully track the changing environment using unlabeled data only. The ability of COMPOSE learning in such a setting comes at a cost: COMPOSE is a relatively computationally expensive algorithm, though not as expensive as APT, at least for the data sets used in our experiments.

The complexity of COMPOSE has been reduced from its original version, where the skeleton algorithm used for compaction was its computationally most expensive module. With the onion-peeling compaction described here, the compaction function is no longer a computational bottle neck—in fact, it is no longer dependent on dimensionality. The most expensive module in COMPOSE is now the α-shape generation, which runs in exponential time with respect to the number of dimensions. We have run some timing experiments, described in the following, to better understand the behavior of the algorithm with respect to its computational complexity.

Fig. 16 shows the computation time, averaged over 50 trials for COMPOSE and five trials for APT, conducted on a modest 2.4-GHz processor (with 6-GB RAM) for each synthetic experiment described in the previous section. In each case, the timing diagrams follow a similar trend: the initial few time steps are computed relatively quickly while a basis of core supports is built up; then, within a few time steps, the

TABLE III
COMPOSE AND APT COMPUTATION COMPARISON

Dataset	COMPOSE [minutes]	APT [minutes]
2D Unimodal	4.16	3,600
2D Multimodal	8.33	20,303
2D Unimodal – Class Added	4.33	21,390
3D Unimodal	26.66	22,776

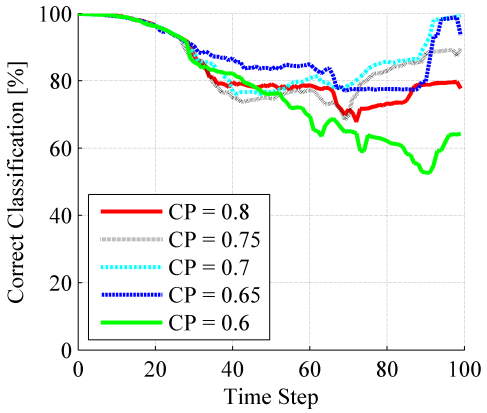

Fig. 17. Typical family of curves with α value ($\alpha = 0.40$ shown) held constant and CP allowed to vary.

algorithm reaches a steady state and maintains approximately the same processing time (per time step) for the remainder of the experiment, unless new classes are added, which then adds a modest additional cost (see change in unimodal Gaussian added-class experiment steady-state computation time at time step 40).

Comparing the unimodal Gaussian experiment (with 100 unlabeled instances added per class, resulting in 200 new instances per time step) and its 2.5 s per time step steady-state processing time with the multimodal Gaussian experiment (with 100 unlabeled instances added for each of the bimodal classes, resulting in 400 new instances per time step) and its 5 s per time step steady-state processing time further shows that COMPOSE runs in nearly linear time with respect to the cardinality of the data.

Comparing the unimodal Gaussian experiment, CP $= 0.70$, with the unimodal added-class experiment, CP $= 0.60$, suggests the greater the CP the faster the algorithm runs, as there are fewer core supports to maintain.

Comparing any of the 2-D experiments to the 3-D experiment shows that the computation time increases greatly with higher dimensional data. This increase in computational complexity with respect to the dimensionality is the primary cost of the current algorithm. We believe the cost is, however, justified given the difficulty of the task the algorithm seeks to solve. We should note that even with the 8-D data, where processing for each time step takes 20–30 min (on a modestly configured computer), COMPOSE is still well within useable limits for many applications that generate data no faster than every 30 min. Any application, for example, that generates hourly or daily data can be easily used with current version of COMPOSE even with higher dimensions. Furthermore, we should reemphasize that the primary bottle neck in COMPOSE is not the data cardinality but rather its dimensionality. Therefore,

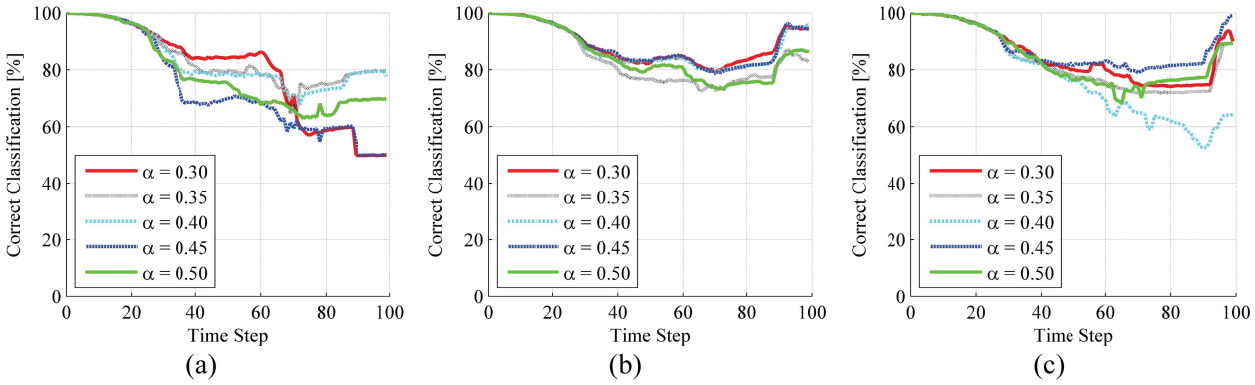

Fig. 18. Family of curves with CP held constant and α value allowed to vary. (a) When CP is too high, e.g., 0.8, or (c) too low, e.g., 0.6, the algorithm is sensitive to variations in α. (b) When CP is selected close to optimal value, e.g., 0.68, the performance variation and the sensitivity to α decrease dramatically.

the algorithm can easily handle large databases with modest dimensionality.

It is also worth noting that all the computation times mentioned above were obtained using a modestly configured computer running an interpreted language (MATLAB). Optimizing the algorithm (many of its steps can be run in parallel), implementing it in a compiled language and running it in a parallel computing setting can further improve its computational efficiency, which is included in our future work.

Comparing computation times of COMPOSE and APT, Table III shows a significant difference. As expensive as COMPOSE is, it completed the synthetic dataset experiments significantly faster than APT on the same computer in the same MATLAB environment.

Finally, because the most expensive module in COMPOSE is the α-shape generation—essentially a density estimation algorithm—alternative density estimation approaches such as GMMs may further improve the computational efficiency. Evaluating such alternative density estimation approaches is also within the scope of our current and future work.

D. Choice of Free Parameters and Their Effects

To better understand the impact of COMPOSE's free parameters, the α value and CP, we have repeated the synthetic data experiments varying each parameter independently. We first looked at the effect of CP, keeping α constant using a family of curves. A sample of these (using the multimodal Gaussian data) is shown in Fig. 17, which shows that a proper choice of CP is necessary. We also plotted performance keeping CP constant and allowing α value to vary—whose sample plots are shown in Fig. 18 for three different values of CP. These results show that when the CP is chosen incorrectly, too high as in Fig. 18(a) or too low as in Fig. 18(c)—the performance varies greatly with respect to α. If CP is, however, chosen properly, as in Fig. 18(b), the algorithm performance becomes less sensitive to the α parameter.

From this preliminary sensitivity analysis, we conclude that CP has a bigger impact on COMPOSE performance. There may be a logical explanation for this: if α shapes are compacted too much, core supports relevant to the future distribution are lost. If compacted too little, core supports may overlap with a rival class misleading the algorithm's direction.

V. Conclusion

We described a new framework, called COMPOSE, for semi-supervised learning of a nonstationary (drifting) environment experiencing extreme verification latency. In this environment, the data arrive in a streaming manner, and beyond an initial batch, the entire data stream is assumed unlabeled. We refer to such an environment as ILSE, where the verification latency is infinite. Our initial results have been quite promising, demonstrating that COMPOSE can indeed learn and track the drifting distributions in such an environment.

COMPOSE can track any drifting environment as long as the class conditional distributions overlap at subsequent time steps. We refer to this condition as limited drift. This is a reasonable assumption, as in most natural phenomena—perhaps with the exception of catastrophic or abrupt failures—changes to the data distribution are usually gradual. One particularly pathological scenario is worth mentioning as an extreme case that violates the limited drift assumption: a sudden change (or switch) of class labels while overall data distribution remains constant. In such a case, there will be no overlap between $p^t(\boldsymbol{x}|y)$ and $p^{t+1}(\boldsymbol{x}|y)$. COMPOSE cannot track such a change, because it receives no future labeled data. Toy examples of this scenario include the shifting hyperplane as used in [9], and rotating checkerboard as used in [4] and [55]. On the other hand, COMPOSE can naturally work in the more relaxed environment, where labeled data are provided regularly or intermittently. In such a case, COMPOSE simply employs the provided labeled data as new core supports to be used in future time steps. COMPOSE can then accommodate the aforementioned change to class membership scenarios, as well as abrupt change scenarios. The experiment that introduced a new class is an example of such a scenario, where COMPOSE was provided with a one-time set of labeled data (only at the time the new class was introduced), and it was able to learn this abrupt change introduced by the new concept (class).

Under the ILSE setting, the focus of this paper, preliminary results show that COMPOSE can closely track the optimal Bayes classifier, while outperforming APT in the regions of class overlap, as well as in scenarios where data distributions diverge into multiple modes. APT requires all the modes to

be presented at initialization and further assumes that any drift in data distribution must be structured or systematic. Such an assumption is not made by COMPOSE. Finally, while COMPOSE is a computationally intensive algorithm, it appears to be more efficient than APT.

Nevertheless, the α-shape construction used by COMPOSE is indeed a computationally expensive process, one that is exponential in dimensionality. Our current and future work includes exploring more efficient ways of constructing α shapes, or using alternate density estimation techniques, such as GMMs or kernel density estimation. While such changes may require modifications to the compaction method, the foundational concepts of COMPOSE remain the same—select instances from the geometric center (core region) of high-density regions of each class to be used as labeled data, and combine them with the unlabeled data of subsequent time step. This is why we refer to COMPOSE more as a framework, rather than just an algorithm. COMPOSE can be a modular family of algorithms, depending on how the core supports are determined, what SSL algorithm is used as a BaseClassifier, or how the compaction is applied.

There is much room for improvement: articulating a more rigorous definition of limited drift (e.g., defining limited drift with respect to Kullback–Leibler divergence or Hellinger distance between two subsequent distributions), optimizing or automating selection of algorithm parameters, and expanding the experimental work to other real-world and even higher dimensional data, all constitute our future work.

Despite its limitations and the aforementioned room for improvement, we believe that the COMPOSE shows a significant promise in addressing extreme verification latency problem, performing quite well against other approaches. It is worth mentioning that COMPOSE's limited drift assumption is much less restrictive than those of other algorithms (including of domain adaptation, even though COMPOSE tackles a more difficult problem). Perhaps most remarkable is the performance comparison of COMPOSE against the Bayes classifier and Learn^{++}.NSE (an ensemble of supervised learners). In these experiments, the experimental conditions for comparison were deliberately set to be grossly unfair against COMPOSE, where the competing algorithms were run in a fully supervised mode.

Finally, we should mention that COMPOSE introduces tools from computational geometry that are not often used in machine learning research but may have applications to other machine learning problem domains. We hope that this paper will stimulate new discussions and new efforts, and perhaps open computational geometry-based approaches to other machine learning problems, where such approaches have been mostly underexplored.

REFERENCES

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

[2] A. Bifet, "Adaptive learning and mining for data streams and frequent patterns," Ph.D. dissertation, Dept. Llenguatges Sistemes Inf., Univ. Politècnica Catalunya, Barcelona, Spain, 2009.

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

[4] R. Elwell and R. Polikar, "Incremental learning of concept drift in non-stationary environments," *IEEE Trans. Neural Netw.*, vol. 22, no. 10, pp. 1517–1531, Oct. 2011.

[5] H. Haibo, C. Sheng, L. Kang, and X. Xin, "Incremental learning from stream data," *IEEE Trans. Neural Netw.*, vol. 22, no. 12, pp. 1901–1914, Dec. 2011.

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

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

[8] P. P. Rodrigues, J. Gama, and J. P. Pedroso, "Hierarchical clustering of time-series data streams," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 5, pp. 615–627, May 2008.

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

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

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

[12] I. Zliobaite, "Combining similarity in time and space for training set formation under concept drift," *Intell. Data Anal.*, vol. 15, no. 4, pp. 589–611, 2011.

[13] G. Marrs, R. Hickey, and M. Black, "The impact of latency on online classification learning with concept drift," in *Knowledge Science, Engineering and Management*, vol. 6291, Y. Bi and M. A. Williams, Eds. Berlin Heidelberg, Germany: Springer-Verlag, 2010, pp. 459–469.

[14] H. Daume and D. Marcu, "Domain adaptation for statistical classiers," *J. Artif. Intell. Res.*, vol. 26, pp. 101–126, Jun. 2006.

[15] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira, "Analysis of representations for domain adaptation," in *Advances in Neural Information Processing Systems*, vol. 19, B. Scholkopf, J. Platt, and T. Hoffman, Eds. Cambridge, MA, USA: MIT Press, 2007, pp. 137–144.

[16] J. P. Sinno and Y. Qiang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[17] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *J. Stat. Planning Inference*, vol. 90, no. 2, pp. 227–244, 2000.

[18] R. Caruana, "Multitask learning," *Mach. Learn.*, vol. 28, no. 41, pp. 41–75, 1997.

[19] J. J. Heckman, "Sample selection bias as a specification error," *Econometrica*, vol. 47, no. 1, pp. 153–161, 1979.

[20] M. Sugiyama, M. Krauledat, and K.-R. Muller, "Covariate shift adaptation by importance weighted cross validation," *J. Mach. Learn. Res.*, vol. 8, pp. 985–1005, May 2007.

[21] J. Huang, A. J. Smola, A. Gretton, K. M. Borgwardt, and B. Scholkopf, "Correcting sample selection bias by unlabeled data," in *Advances in Neural Information Processing Systems*, vol. 19, B. Scholkoph, J. Platt, and T. Hoffman, Eds. Cambridge, MA, USA: MIT Press, 2007, pp. 601–608.

[22] J. Blitzer, R. McDonald, and F. Pereira, "Domain adaptation with structural correspondence learning," in *Proc. Conf. EMNLP*, 2006, pp. 120–128.

[23] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, "A theory of learning from different domains," *Mach. Learn.*, vol. 79, nos. 1–2, pp. 151–175, Oct. 2009.

[24] A. Margolis, "Automatic annotation of spoken language using out-of-domain resources and domain adaptation," Ph.D. dissertation, Dept. Comput. Sci., Univ. Washington, Seattle, WA, USA, 2011.

[25] O. Chapelle, B. Scholkopf, and A. Zien, *Semi-Supervised Learning*. Cambridge, MA, USA: MIT Press, 2006.

[26] K. Chen and S. Wang, "Semi-supervised learning via regularized boosting working on multiple semi-supervised assumptions," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 129–143, Jan. 2011.

[27] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. R. Stat. Soc. Ser. B, Methodol.*, vol. 39, no. 1, pp. 1–38, 1977.

[28] X. Zhu and A. Goldberg, *Introduction to Semi-Supervised Learning*. San Rafael, CA, USA: Morgan & Claypool, 2009, pp. 31–32.

[29] T. Joachims, "Transductive inference for text classification using support vector machines," in *Proc. 16th Int. Conf. Mach. Learn.*, 1999, pp. 200–209.
[30] V. Vapnik and A. Sterin, "On structural risk minimization for overall risk in a problem of pattern recognition," *Autom. Remote Control*, vol. 10, no. 3, pp. 1495–1503, 1977.
[31] D. Zhou, O. Bousquet, T. Lal, J. Weston, and B. Scholkopf, "Learning with local and global consistency," in *Advances in Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 2004, pp. 321–328.
[32] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CALD-02-107, 2002.
[33] M. Masud, J. Gao, L. Khan, J. Han, and X. Li, "Classification and novel class detection in data streams with active mining," in *Proc. Pacific-Asia Conf. Knowl. Discovery Data Mining*, 2010, pp. 311–324.
[34] A. Goldberg, X. Zhu, A. Furger, and J. Xu, "OASIS: Online active semi-supervised learning," in *Proc. 25th Conf. Artif. Intell.*, 2011, pp. 1–6.
[35] I. Zliobaite, A. Bifet, B. Pfahringer, and G. Holmes, "Active learning with evolving streaming data," in *Machine Learning and Knowledge Discovery in Databases*, vol. 6913. Berlin Heidelberg, Germany: Springer-Verlag, 2011, pp. 597–612.
[36] R. Capo, K. Dyer, and R. Polikar, "Active learning in nonstationary environments," in *Proc. IJCNN*, Houston, TX, USA, 2013.
[37] A. Kuh, T. Petsche, and R. L. Rivest, "Learning time-varying concepts," in *Advances in Neural Information Processing Systems*, Denver, CO, USA: MIT Press, 1990, pp. 183–189.
[38] L. L. Minku, A. P. White, and X. Yao, "The impact of diversity on online ensemble learning in the presence of concept drift," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 5, pp. 730–742, May 2010.
[39] A. Goldberg, M. Li, and X. Zhu, "Online manifold regularization: A new learning setting and empirical study," *Computer*, vol. 5211, no. 1, pp. 393–407, 2008.
[40] P. Li, X. Wu, and X. Hu, "Mining recurring concept drifts with limited labeled streaming data," in *Proc. 2nd Asian Conf. Mach. Learn.*, Tokyo, Japan, 2010, pp. 241–252.
[41] G. Ditzler and R. Polikar, "Semi-supervised learning in nonstationary environments," in *Proc. Int. Joint Conf. Neural Netw.*, 2011, pp. 2741–2748.
[42] M. Masud, J. Gao, L. Khan, and J. Han, "A practical approach to classify evolving data streams: Training with limited amount of labeled data," in *Proc. IEEE 8th Int. Conf. Data Mining*, Washington, DC, USA, Dec. 2008, pp. 929–934.
[43] P. Zhang, X. Zhu, and L. Guo, "Mining data streams with labeled and unlabeled training examples," in *Proc. IEEE 9th Int. Conf. Data Mining*, Miami, FL, USA, Dec. 2009, pp. 627–636.
[44] I. Katakis, G. Tsoumakas, and I. Vlahavas, "Tracking recurring contexts using ensemble classifiers: An application to email filtering," *Knowl. Inf. Syst.*, vol. 22, no. 3, pp. 371–391, 2010.
[45] P. Zhang, X. Zhu, J. Tan, and L. Guo, "Classifier and cluster ensembles for mining concept drifting data streams," in *Proc. IEEE 10th Int. Conf. Data Mining*, Dec. 2010, pp. 1175–1180.
[46] G. Krempl and V. Hofer, "Classification in presence of drift and latency," in *Proc. IEEE Int. Conf. Data Mining Workshops*, Dec. 2011, pp. 596–603.
[47] R. Alaiz-Rodriguez, A. Guerrero-Curieses, and J. Cid-Sueiro, "Class and subclass probability re-estimation to adapt a classifier in the presence of concept drift," *Neurocomputing*, vol. 74, no. 16, pp. 2614–2623, Sept.2011.
[48] V. Hofer and G. Krempl, "Drift mining in data: A framework for addressing drift in classification," *Comput. Stat. Data Anal.*, vol. 57, no. 1, pp. 377–391, Jan. 2013.
[49] G. Krempl, "The algorithm APT to classify in concurrence of latency and drift," in *Advances in Intelligent Data Analysis* (Lecture Notes in Computer Science), vol. 7014, J. Gama, E. Bradley, and J. Hollmen, Eds. Berlin Heidelberg, Germany: Springer-Verlag, 2011, pp. 222–233.
[50] K. Dyer and R. Polikar, "Semi-supervised learning in initially labeled non-stationary environments with gradual drift," in *Proc. IJCNN*, 2012, pp. 1–9.
[51] M. Teichmann and M. Capps, "Surface reconstruction with anisotropic density-scaled alpha shapes," in *Proc. IEEE Visualizat.*, Oct. 1998, pp. 67–72.
[52] C. Barber, D. Dobkin, and H. Huhdanpaa, "The QuickHull algorithm for convex hulls," *ACM Trans. Math. Softw.*, vol. 22, no. 4, pp. 469–483, Dec. 1996.
[53] D. Pedoe, *Circles: A Mathematical View*, 2nd ed. New York, NY, USA: Dover, 1979.
[54] H. Edelsbrunner and E. Mucke, "Three-dimensional alpha shapes," *ACM Trans. Graph.*, vol. 13, no. 1, pp. 43–72, 1994.
[55] R. Elwell and R. Polikar, "Incremental learning in nonstationary environments with controlled forgetting," in *Proc. IJCNN*, 2009, pp. 771–778.

Karl B. Dyer received the B.S. degree in mechanical engineering from Rowan University, Glassboro, NJ, USA, in 2010, where he is currently pursuing the M.S. degree in electrical and computer engineering.

His current research interests include concept drift, nonstationary environments, semi-supervised learning, and applications of machine learning in robotics and automation.

Robert Capo received the B.S. degree in electrical and computer engineering from Rowan University, Glassboro, NJ, USA, in 2013, where he is currently pursuing the M.Sc. degree in machine learning.

He has conducted and published research on nonstationary, semi-supervised, and active learning environments. His current research interests include neuroscience and its application to machine learning.

Robi Polikar (S'93–M'00–SM'08) received the B.Sc. degree in electronics and communications engineering from Istanbul Technical University, Istanbul, Turkey, in 1993, and the M.Sc. and Ph.D. degrees in electrical engineering and biomedical engineering from Iowa State University, Ames, IA, USA, in 1995 and 2000, respectively.

He is a Professor of electrical and computer engineering with Rowan University, Glassboro, NJ, USA. His current research interests include computational intelligence including ensemble systems, incremental and nonstationary learning, and various applications of pattern recognition in bioinformatics and biomedical engineering.

Dr. Polikar is a member of 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 Programs. He is also an Associate Editor of IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS.