Learning Under Extreme Verification Latency Quickly: FAST COMPOSE

Muhammad Umer
Rowan University
umerm5@students.rowan.edu

Christopher Frederickson
Rowan University
fredericc0@students.rowan.edu

Robi Polikar
Rowan University
polikar@rowan.edu

***Abstract*—One of the more challenging real-world problems in computational intelligence is to learn from non-stationary streaming data, also known as concept drift. Perhaps even a more challenging version of this scenario is when – following a small set of initial labeled data – the data stream consists of unlabeled data only. Such a scenario is typically referred to as learning in initially labeled nonstationary environment, or simply as extreme verification latency (EVL). In our prior work, we described a framework, called COMPOSE (COMPacted Object Sample Extraction) that works well in this type of environment, provided that the data distributions experience limited drift. The central premise behind COMPOSE is core support extraction, in which α-shapes or density estimation is used to extract the most representative instances – the core supports that typically lie in the center of the feature space for each class – to be used as labeled data in future time-steps. This process, however, is computationally very expensive especially for high dimensional data. In this paper, we describe a modification to COMPOSE that allows the algorithm to work without core support extraction. We call the new algorithm FAST COMPOSE. Several datasets are used to compare the performance of FAST COMPOSE with the original COMPOSE, as well as with SCARGC (another algorithm that can address EVL), both in accuracy and in execution time. The results obtained show the promising potential of using FAST COMPOSE.**

I. Introduction

Most machine learning algorithms are based on the basic assumption that data are drawn from a fixed but unknown distribution. This assumption implies that test or field data come from the same distribution as the training data. In reality, this assumption simply does not hold in many real world problems that generate data whose underlying distributions change over time. Such scenarios render most traditional learning algorithms ineffective at best, misleading and inaccurate at worst.

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 of the applications of the concept drift problem. The concept drift problem poses great difficulty because streaming data are usually unlabeled and unstructured. Much of the prior work addressing the concept drift problem focuses on using supervised approaches, for example [1] [2]. However, an algorithm that is able to handle scenarios where labeled information is very scarce (or even no longer available after an initial step) would be more beneficial.

Several different types of approaches have recently been proposed in the literature. One group of approaches that are designed to handle the disparate training and test data distributions is the so-called domain adaptation approaches [3] [4], where the training data distributions (called the source domain) and test data distributions (called the target domain) are considered to be different but related. Typically, in domain adaptation approaches, the classification model is trained with a modified version of the original training data that is weighted, through an approach called instance weighting, such that the source domain training data distribution behaves more like the target domain test data distribution. However, domain adaptation approaches typically assume that labeled data are available in abundance in the source domain and that both the source and target domains share the same support. Furthermore, domain adaptation approaches only consider a single time-step scenarios and are not equipped for, nor are they designed to handle, streaming data. Therefore domain adaptation approaches are not well suited for concept drift problems where data are generated in a streaming or continuous fashion whose distribution may change over time.

Ensemble based approaches such as Learn^{++}.NSE [1], Learn^{++}.NIE [5], DWM [2], and SEA [6] represent a different family of approaches to tackle concept drift problems. All of these approaches take into account the difference in the probability distribution of the training (source) and test (target) data distribution, i.e., $p_s(x,y) \neq p_t(x,y)$ where p_s and p_t are source and target distributions, respectively for the features x and labels y, respectively. Unlike domain adaptation approaches, these ensemble based approaches are indeed capable of tracking data distributions over a streaming setting. However, ensemble approaches also require a large amount of labeled data, and the potential scarcity or the high cost of obtaining labeled data is a major obstacle in using these approaches. In an effort to reduce the amount of required labeled data, semi supervised learning (SSL) approaches have also been used in this scenario where a hypothesis is formed using modest amount of labeled data and more abundant unlabeled data. The primary application domain of SSL techniques has been in stationary environments, but recently the focus in this area has shifted towards non stationary distributions. SSL approaches, of course, also require labeled data at each

978-1-5090-4240-1/16/$31.00 ©2016 IEEE

time-step [7], albeit in smaller quantities. Active learning (AL) is another technique used to combat the limited availability of labeled data in general and in concept drift problems in particular [8]. In active learning the learner actively chooses which data instances – if labeled – would provide the most benefit. The goal in AL algorithms is therefore to find the minimum number of labeled examples that provide the maximum benefit. This is most commonly achieved by assuming that there is an oracle or expert that can be queried for the labels of any example on demand. Active learning approaches cannot function, however, if the requested labels cannot be provided on demand, a potentially restricting limitation.

More recent research, typically referenced as verification latency, acknowledges an additional and important constraint that must be addressed: labeled data may not be available at every time-step, nor even in regular intervals, which significantly complicates the learning process. Verification latency, as denoted in [9], 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. In the *extreme verification latency* scenario, this lag becomes infinite, meaning that no labeled data are ever received after initialization. We call such an environment as an initially labeled non stationary environment (ILNSE) or simply initially labeled streaming environment (ILSE) [10]. In our prior work, we described an algorithm called COMPOSE [10] (**COMP**acted **O**bject **S**ample **E**xtraction) that can learn in an ILSE or extreme verification latency setting. COMPOSE works remarkably well and makes no assumption on the nature of the distribution other than requiring limited drift (a common assumption of all concept drift algorithms). However, the ability of COMPOSE to track a nonstationary environment in an extreme verification latency scenario comes at a steep cost: COMPOSE is computationally expensive, very expensive. Specifically, COMPOSE's computational complexity is exponential in dimensionality (but only linear in cardinality). The primary motivation for the work described in this paper is to explore how COMPOSE can be made more efficient with little or no loss on its classification performance.

II. Related Work

The extreme verification latency setting requires an entirely different framework and approach from the traditional supervised learning because it requires class information to be propagated forward through not just several time-steps, but of indefinite duration of only unlabeled data. To the best of our knowledge, we are aware of only four algorithms that are designed to address the initially labeled non-stationary environment or extreme verification latency. Arbitrary subpopulations tracker (APT) [11], stream classification algorithm guided by clustering (SCARGC) [12], micro-clusters for classification (Mclassification) [13], and **COMP**acted **O**bject **S**ample **E**xtraction (COMPOSE) [10]. In this section we briefly describe the central premises of these algorithms.

APT is based on the principle that each class in the data can be represented as a mixture of arbitrarily distributed subpopulations. The APT algorithm, similar to all concept drift approaches, makes the gradual drift assumption, i.e., that the drift within the dataset is limited. However, the APT algorithm also requires that i) the drift can be represented as a piecewise linear function, ii) the covariance of each subpopulation remains constant where sub-population is defined as a mode in the class conditional distribution, iii) each subpopulation to be tracked must be present at the initialization, iv) the drift remains constant, and v) the drift only affects the conditional feature distribution $P(X|Z)$. The conditional posterior distribution $P(Y|Z)$ and prior distribution of components $P(Z)$ remains fixed, where $P(X)$ represents the feature distribution, $P(Y)$ represents the distribution of the class labels and $P(Z)$ represents the mixing proportions, i.e., component prior distributions. The algorithm also assumes that the population parameters, bandwidth matrix and exemplar offsets representing each subpopulation via non-parametric density estimation, remain constant and attempts to solve for the drift parameters, change in centroid between each time-step and starting position.

The learning strategy of APT is twofold; first, the optimal one-to-one assignment between labeled instances in time-step t and unlabeled instances in time-step $t+1$ is determined using the expectation maximization algorithm following the assumption that $P(Z)$ remains static. The expectation maximization algorithm begins by predicting which instances are most likely to correspond to a given sub-population in the expectation step. During the maximization step, the algorithm determines which drift parameters maximize the expectation. Then, the classifier is updated to reflect the population parameters of the newly received data and drift parameter relating the previous time-step to the current one. When the assumptions are satisfied, APT works very well. However, APT has two primary weaknesses: 1) the many assumptions made by APT often do not hold true, causing a decrease in performance, and 2) the algorithm is computationally very expensive, significantly more so than even the original COMPOSE, as shown by the tests performed in [10].

SCARGC operates by repeatedly clustering unlabeled input data, and then classifying the clusters using the labeled clusters from the last time-step. A fixed number of unlabeled examples are stored in a pool in the classification phase, and the unlabeled instances are then clustered into k clusters. Current and past cluster centroids are used to track the drifting classes over time. In other words, the mapping between clusters is performed by centroid similarity between current and previous iterations using Euclidean distance. Given the current centroids from the most recent unlabeled clusters and past centroids from the previously labeled clusters, one-nearest-neighbor algorithm or a support vector machine is used to label the centroid from current unlabeled clusters. This algorithm is also based on the assumptions that the drift is gradual or

incremental, the number of classes is known ahead of time and is constant over time. SCARGC is computationally efficient, but its performance is highly dependent on the clustering phase. Additionally, the need for prior knowledge such as the number of classes, and the number of modes for each class in the data, limits the utility of this approach when such information is not available.

Mclassification uses the idea of micro clusters (MC) [14] to adapt to the changes in the data over time. Micro clusters is a method of storing information about a set of data points without requiring all examples to be retained. A set of labeled MC are first built from the initial labeled data. At each time-step, each example from the streaming input data receives its label from the nearest MC. If the addition of an example in its corresponding nearest MC causes its MC radius not to exceed the maximum MC radius defined by the user, this example is added to the nearest MC and its sufficient statistics, i.e., its centroid and micro cluster radius are updated; otherwise a new MC is created. MClassification does not require the number of clusters to be known prior to execution as with APT or SCARGC, however it is also computationally expensive.

We previously proposed COMPOSE, which originally used a generalization of convex hull called α-shapes to deal with incremental drifts and infinitely delayed labels. The α-shapes are geometrical constructs obtained from data that are then compacted (shrunk) to determine those current data points that are most likely to represent the distribution at the next time-step. These instances are called core supports and serve as labeled instances during the semi-supervised learning at the next time-step. The high computational cost of the α-shape construction and the requirement of the tuning of its two critical parameters, i.e., α and compaction proportion (CP) are the main weaknesses of the original COMPOSE algorithm. Recently, we have replaced the α-shape construction with a Gaussian mixture model based density estimation approach to determine the core supports, which significantly reduced the computational burden of the algorithm [15], though the algorithm still remained fairly computationally expensive. In this contribution, we describe a simplification of COMPOSE, which we refer to as FAST COMPOSE, which eliminates the core support extraction entirely and dramatically improves the computational efficiency of the algorithm with little or no performance degradation, and in fact provide even some performance improvement in certain cases.

III. Approach

The **COMP**acted **O**bject **S**ample **E**xtraction (COMPOSE) framework was introduced in [10] to address the extreme verification latency problem in an ILSE setting, i.e., learn drifting concepts from a streaming non stationary environment that provides only unlabeled data after initialization. COMPOSE originally consisted of three primary steps: 1) combine initial labels with new unlabeled data to train a semi supervised learning (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 those instances, called core supports, that fall into the compacted region, which itself 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. COMPOSE is really a framework, rather than just an algorithm, as it is very flexible, versatile and modular: any number of core support extraction (CSE) routines using any density estimation techniques such as Gaussian mixture models (GMM) or k-nearest neighbor can be used in place of α shapes, and any SSL algorithm can be used that the user believes to match the characteristics of the data to improve the performance of the algorithm [15]. The pseudocode and implementation details of COMPOSE can be seen in Algorithm 1.

In its current form, COMPOSE takes the following inputs: an SSL algorithm such as cluster and label, label propagation [16], or semi-supervised support vector machines [17] with relevant free parameters; a CSE algorithm such as α shape, Gaussian mixture model (GMM) or k-nearest neighbor (kNN); and a compaction percentage, CP, that represents the percentage of current labeled instances to use as core supports. The algorithm begins by receiving M initially labeled instances, L^0, and corresponding labels, Y^0, of C classes in step 1. For each time-step, a new set of N unlabeled instances, U^t, is received. The SSL algorithm is then executed given the current unlabeled and labeled instances and returns a hypothesis h^t that classifies all unlabeled instances of the current time-step in step 4. The hypothesis is then used to generate a combined set of data, D^t, in step 5, and the combined data for each class is used as the input for the CSE routine in step 8. The resulting core supports CS_c, for each class c, are appended to be used as current labeled data in the next time-step in step 9.

A number of issues have been identified with the current implementation of COMPOSE. Originally, it was thought that the core support extraction routine would extract a region that has a high probability of overlap with drifted unlabeled data at the next time-step. However, because the core support extraction routine is executed using the labeled data at time-step t (which themselves are the core supports extracted at time-step $t-1$), and the hypothesis at time-step t, the most dense region will lie in the overlap between the labeled and unlabeled data from time-step t. Therefore, the core supports at time-step t will be further away from the unlabeled data at time-step $t+1$, and in turn the CSE routine will extract a region that has a lower probability of overlap with the drifted data. Figure 1 illustrates the above described scenario, where the blue circle represents the labeled data at time-step t (i.e., the core supports extracted from time-step $t-1$) denoted as CS^{t-1}/L^t; the left gray circle represents the unlabeled data or hypothesis at time-step t denoted as U^t/h^t, the right gray circle represents the unlabeled data at time-step $t+1$ denoted as U^{t+1}/h^{t+1}, and the pink circle represents the core supports extracted in time-step t to be used as labeled information for

Algorithm 1 COMPOSE

Inputs: SSL algorithm - **SSL** with relevant free parameters; CSE algorithm - **CSE**; Compaction percentage - **CP**

1: Receive labeled data
$L^0 = \{x_l^t \in X\}$,
$Y^0 = \{y_l^t \in Y = \{1, \ldots, C\}, l = 1, \ldots, M\}$
2: **for** $t = 0, 1, ...$ **do**
3: Receive unlabeled data $U^t = \{x_u^t \in X\ ,\ u = 1, ..., N\}$
4: Call **SSL** with L^t , Y^t, and U^t
to obtain hypothesis, $h^t : \mathrm{X} \rightarrow \mathrm{Y}$
5: Let $D^t = \{(x_l^t, y_l^t) : x \in L^t \forall l\} \cup$
$\{(x_u^t, h^t(x_u^t)) : x \in U^t \forall u\}$
6: Set $L^{t+1} = \emptyset, Y^{t+1} = \emptyset$
7: **for** each class $c = 1, 2, ..., C$ **do**
8: Call **CSE** with CP and D_c^t
to extract core supports, CS_c
9: Add core supports to labeled data
$L^{t+1} = L^{t+1} \cup CS_c$
$Y^{t+1} = Y^{t+1} \cup \{y_u : u \in [|CS_c|], y = c\}$
10: **end for**
11: **end for**

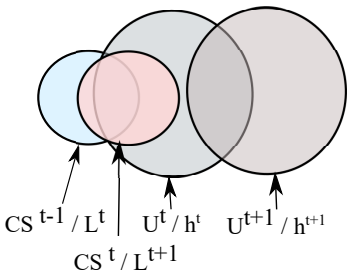


Fig. 1. Example depicting core support extraction procedure in COMPOSE

time-step $t + 1$. We see that the unlabeled data at time-step $t+1$ is far from the labeled data using this implementation of core support extraction.

In [18], we provided a formal definition of "limited drift," and how that differs from "gradual drift." Then, we also showed that COMPOSE can work equally well for scenarios when there is no significant overlap as long as the distance between the unlabeled data with core supports of the same class is less than the distance from the nearest core supports from any other class. We refer to this condition as limited drift and distinguish it from gradual drift that requires an overlap of distributions in subsequent time-steps. As a result, we showed that the condition of significant overlap (or "gradual drift") can be completely eliminated, and replaced with the more relaxed condition of limited drift. In cases where there is no significant overlap, the core support extraction procedure has very little impact on accuracy as it does not significantly impact the centroid of each class. When there is no significant overlap, the SSL algorithms, in particular cluster and label, classify the unlabeled data using the nearest centroid, and therefore removing CSE is expected to have little impact on accuracy.

Additionally, the density estimation procedure is impractical for high dimensional data due to its computational complexity. In this paper, the core support extraction procedure of COMPOSE is removed, and all instances labeled by the semi-supervised algorithm are then used as core supports, i.e., the most representative instances for the future time-steps. We call this modified version of the algorithm FAST COMPOSE.

FAST COMPOSE works as shown in Algorithm 2. The only input for FAST COMPOSE is a choice of an SSL algorithm with its relevant free parameters. As with COMPOSE, the algorithm begins by receiving M initially labeled instances, L^0, and corresponding labels Y^0, of C classes in step 1. For each time-step, a new set of N unlabeled instances U^t is received. The SSL algorithm is then executed given the current unlabeled and labeled instances to receive the hypothesis h^t of the current time-step in step 4. The hypothesis is then used to label the data for the next time-step in steps 5 - 8.

Algorithm 2 FAST COMPOSE

Input: SSL algorithm - **SSL** with relevant free parameters

1: Receive labeled data
$L^0 = \{x_l^t \in X\}$,
$Y^0 = \{y_l^t \in Y = \{1, \ldots, C\}, l = 1, \ldots, M\}$
2: **for** $t = 0, 1, ...$ **do**
3: Receive unlabeled data $U^t = \{x_u^t \in X\ ,\ u = 1, ..., N\}$
4: Call **SSL** with L^t , Y^t, and U^t
to obtain hypothesis, $h^t : \mathrm{X} \rightarrow \mathrm{Y}$
5: Let $D^t = \{(x_u^t, h^t(x_u^t)) : x \in U^t \forall u\}$
6: Set $L^{t+1} = \emptyset, Y^{t+1} = \emptyset$
7: **for** each class $c = 1, 2, ..., C$ **do**
8: $CS_c = \{x : x \in D_c^t\}$, and add to labeled data for next time-step
$L^{t+1} = L^{t+1} \cup CS_c$
$Y^{t+1} = Y^{t+1} \cup \{y_u : u \in [|CS_c|], y = c\}$
9: **end for**
10: **end for**

Figure 2 summarizes the difference in the working flow of both algorithms where w/ CSE indicates "with core support extraction," i.e., the original COMPOSE, and w/o CSE indicates COMPOSE run without core support extraction, which is FAST COMPOSE.

IV. Experiments and Results

A. Experimental Setup

Sixteen synthetic datasets and one real dataset are used to evaluate FAST COMPOSE; some of which were originally provided by us in our prior works of [5] and [10], and others provided by the authors of SCARGC in [12]. We compare the accuracy and execution time of FAST COMPOSE with COMPOSE using α shapes and Gaussian mixture models (GMM) as core supports extraction procedures, and also with SCARGC with one nearest neighbor (1-NN) and support vector machines (SVM) as classifiers on the benchmark datasets. APT and MClassification are not included in this analysis due to their prohibitively high computational cost, given that our primary goal in this effort is reducing the computational complexity.

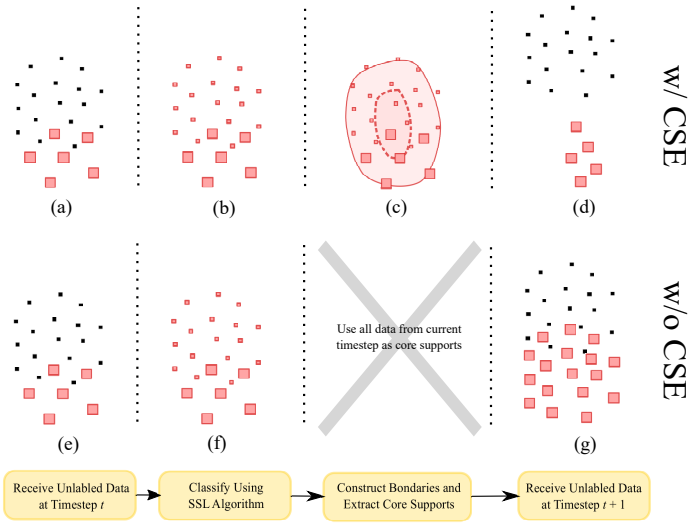


Fig. 2. Difference in working flow of COMPOSE and FAST COMPOSE

TABLE I
DATASETS DESCRIPTION

Datasets	Number of classes	Number of features	Cardinality	Drift interval
1CDT	2	2	16000	400
1CHT	2	2	16000	400
1CSurr	2	2	55283	600
2CDT	2	2	16000	400
2CHT	2	2	16000	400
4CE1CF	5	2	173250	750
4CR	4	2	144400	400
4CRE-V1	4	2	125000	1000
4CRE-V2	4	2	183000	1000
5CVT	5	2	40000	1000
FG_2C_2D*	2	2	200000	2000
GEARS_2C_2D	2	2	200000	2000
MG_2C_2D*	2	2	200000	2000
UG_2C_2D*	2	2	100000	1000
UG_2C_3D*	2	3	200000	2000
UG_2C_5D*	2	5	200000	2000
keystroke	4	10	1600	200

All algorithms are evaluated 500 times on each dataset on a server with 2 x Intel Xeon E5-2670v3 processors and 64 GB DDR4 memory, the results of which are then averaged. As the purpose of this analysis is primarily to find the fastest algorithm, executions that take over 500 seconds are aborted and given a rank of last place.

The datasets tested in these experiments contain concept drift either synthetically (i.e., the dataset is deliberately designed to have a concept drift, by drawing from a drifting distribution) or due to the process used to generate the dataset. The one real dataset, keystroke, attempts to classify between four users based on typing patterns. The data is generated by having each user type a phrase repeatedly. It is expected that the users will become faster at typing the phrase over time and therefore induce concept drift. Table I summarizes the properties of all the datasets, which includes number of classes, number of features (dimensionality), the number of instances (cardinality), and drift interval. The drift interval is used as the batch size, and the number of time-steps for each example is simply the cardinality divided by the drift interval.

The datasets marked with an asterisk were originally developed and provided by us, whereas others are developed and provided by authors of SCARGC as described in [12]. All datasets are graciously provided by authors of SCARGC at one convenient web page for the use of the machine learning community, and can be found in [19]. This web page also provides additional information about each of the datasets, including videos that illustrate the drift in each synthetic dataset as well as the explanations of the convention used for naming the datasets.

B. Results

Table II shows the average accuracy and Table III shows the average execution time (in seconds) of all algorithms evaluated on all datasets. In order to determine whether there is statistically significant difference among the algorithms in terms of accuracy and execution times as evaluated over all datasets, we used the Friedman test and its corresponding Nemenyi post-hoc test. The post-hoc test results comparing the statistical significance ($p \leq 0.05$) for accuracy and execution time are found in Tables IV and V, respectively. Empty cells at the intersection of any two algorithms indicate that the difference in accuracy or execution time between those two algorithms was not statistically significant. A left arrow ($\leftarrow$) or an up arrow ($\uparrow$) at the intersection of any two algorithms represents a statistically significant difference, with the direction of the arrow indicating the classifier that performed significantly better.

It is important to note that both the accuracy and execution times of SCARGC differ – albeit slightly – from those reported in [12] for the same datasets. These experiments were run on different computer hardware which may be optimized for a different workload. Additionally, when batching drifting time series data for analysis, there is a trade-off between the cardinality of the dataset at each time-step and the amount of drift within the batch that must be considered. A smaller batch size allows for faster adaptation to concept drift, however can also result in lack of (sufficient) data for a particular class [12]. COMPOSE avoids this dilemma by working only with already batched time series data. SCARGC treats the batch size as a free parameter called the *pool size*. In these experiments, both the batch size for COMPOSE and the pool size for SCARGC were set at the drift interval from Table I.

We expected FAST COMPOSE to perform better than all other other algorithms in terms of execution time, and this was in indeed the case. We did not know what to expect about the accuracy performance of FAST COMPOSE, and were rather pleasantly surprised that on most datasets it actually performed better, providing the lowest rank (lower rank is better in terms of performance, rank 1 is the best algorithm and rank 5 is the worst algorithm) when averaged across all datasets.

While FAST COMPOSE's accuracy was better than that

of other algorithms when averaged across all datasets, the differences were not statistically significant when compared to COMPOSE using either GMMs or α shapes as seen in Table IV. On the other hand, both FAST COMPOSE and COMPOSE with GMM showed a statistically significant improvement over both SCARGC 1-NN and SCARGC SVM. In the datasets tested, the simplifications made to COMPOSE have had no negative impact on accuracy.

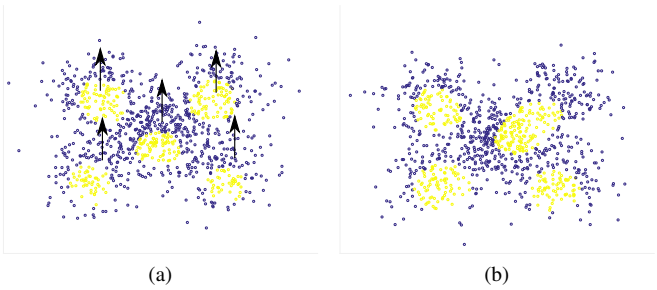
With respect to execution time, as expected, FAST COMPOSE showed a statistically significant improvement over COMPOSE using α shapes, SCARGC 1-NN, and SCARGC SVM. The last one is noteworthy, since SCARGC SVM was previously known as the fastest algorithm in literature per claims made in [12]. SCARGC SVM and 1-NN were slightly faster than FAST COMPOSE in execution time only for the keystroke dataset, which can be attributed to the specific characteristics of this dataset. Specifically, the keystroke dataset contains far fewer time-steps as compared to all synthetic datasets. In the first time-step, the cluster and label implementations of SSL has an additional overhead, as it performs a number of replications in order to improve the reliability. For a large number of time-steps, the impact of the longer first time-step calculations become negligible, whereas the impact is more significant on datasets with fewer time-steps. These results provide evidence that FAST COMPOSE is a very effective approach both in terms of classification accuracy and execution time.

C. Detailed Comparison of FAST COMPOSE and COMPOSE

While COMPOSE and FAST COMPOSE provide similar classification performance (in terms of statistical significance), looking at a particular dataset where the difference was significant may provide some additional insight. Table II listing the average accuracy of these two algorithms shows that FAST COMPOSE and COMPOSE perform similarly for all benchmark datasets except 5CVT. 5CVT is a dataset in which five classes (5C) are translating vertically (VT) in one direction. FAST COMPOSE performs dramatically better compared to COMPOSE (as well as all other algorithms) showing that for this dataset extracting the core supports of the most dense region by using α-shape construction or any density estimation procedures (as performed by COMPOSE) fails to provide the most representative instances, while using all instances as labeled information helps significantly in improving the performance.

The reason COMPOSE underperforms on this dataset is because applying core support extraction introduces a bias in the decision boundary away from the unlabeled data at the next time-step as shown in Figure 1. Ultimately, this bias makes it difficult for the semi-supervised learning (SSL) algorithm to cluster the data properly. When using all instances from the previous time-step as the labeled information, the labeled information is closer to the unlabeled data and this biasing does not occur.

Figure 3 shows the core supports extracted from 5CVT using core support extraction. In subfigure (a), the core supports extracted are biased slightly toward the lower half of the clusters due to the density estimation procedure. This bias, in turn, causes the misclassification of an entire class in subfigure (b). By removing core support extraction and treating all data as core supports, some portion of the labeled data will always be either as close or closer to the cluster of unlabeled data.



(a) (b)

Fig. 3. 5CVT core supports extracted using Gaussian mixture models at: (a) timestep 2 where yellow are core supports and blue are unlabeled data, the core supports are biased toward the lower half of each cluster, the motion of the cluster is indicated by the arrows; (b) time-step 3, due to the bias from the last timestep, one cluster was misclassified.

V. Conclusion & Future work

In this paper, we introduced a modification to COMPOSE that significantly increased the execution speed of the algorithm. The modified version of COMPOSE, named FAST COMPOSE, is the original COMPOSE whose core support extraction step is replaced by using all of the instances labeled by the SSL in the previous time-step as the new labeled data to be used for the next time-step's SSL step. This simplification of the algorithm produce improved results in both classification accuracy and execution time.

Further work is needed to generate or acquire more challenging datasets as most algorithms perform similarly on the current synthetic benchmark datasets. Currently, there is a lack of datasets that contain abruptly changing distributions, datasets with recurring concepts or class imbalances, datasets that have substantial feature or class noise, datasets with significant amount of outliers, datasets with very little or almost no shared support, and high dimensional datasets to name a few. Often, it is a challenging dataset, or a collection of datasets that provide the motivation for the development of specialized algorithms within a specific disciple. Additionally, future work includes comparing the strengths and weaknesses of different algorithms in the literature that operate under extreme verification latency to provide a fair comparison of the existing techniques.

Acknowledgment

This material is based upon work supported by the National Science Foundation under grant no. 1310496 and grant no. 1429467. Graduate student Muhammad Umer and undergraduate student Christopher Frederickson contributed equally to this work.

TABLE II
AVERAGE ACCURACY

DATASETS	COMPOSE(α shape)	COMPOSE (GMM)	FAST COMPOSE	SCARGC(1-NN)	SCARGC(SVM)
1CDT	99.9615 (2.5)	99.9615 (2.5)	**99.9679** (1)	99.7949 (4)	99.6097 (5)
1CHT	99.6026 (2)	**99.6154** (1)	99.5705 (3)	99.3974 (4)	99.2641 (5)
1CSurr	90.9524 (4)	84.6515 (5)	**95.641** (1)	94.5717 (3)	94.998 (2)
2CDT	96.5769 (2)	**96.609** (1)	95.1731 (3)	88.0513 (4)	87.8301 (5)
2CHT	90.391 (2)	**90.4423** (1)	89.4103 (3)	85.9551 (4)	85.859 (5)
4CE1CF	93.9204 (3)	93.9012 (4)	93.9528 (2)	**94.0261** (1)	92.7944 (5)
4CR	99.9924 (2)	99.9924 (2)	99.9924 (2)	99.9604 (4)	98.943 (5)
4CRE-V1	80.7883 (3)	**98.5887** (1)	97.7484 (2)	74.9371 (5)	74.9578 (4)
4CRE-V2	**92.5907** (1)	92.5857 (2)	92.4632 (3)	91.3998 (5)	91.4808 (4)
5CVT	57.9676 (2)	49.9304 (3)	**81.3348** (1)	46.2087 (4)	45.9475 (5)
FG_2C_2D	87.897 (4)	95.5454 (3)	95.5813 (2)	timeout (5)	**95.585** (1)
GEARS_2C_2D	90.9792 (2)	82.2135 (3)	**91.2637** (1)	timeout (5)	74.915 (4)
MG_2C_2D	**93.1162** (1)	92.0298 (4)	93.0167 (2)	timeout (5)	92.9229 (3)
UG_2C_2D	95.6347 (2)	**95.6378** (1)	95.6112 (3)	95.4439 (4)	95.4368 (5)
UG_2C_3D	timeout (4.5)	**95.198** (1)	95.1202 (2)	timeout (4.5)	94.8785 (3)
UG_2C_5D	timeout (4.5)	82.9258 (3)	**91.9889** (1)	timeout (4.5)	91.1504 (2)
keystroke	timeout (5)	**87.4921** (1)	85.9204 (3)	85.6637 (4)	86.3411 (2)
Average Rank (lower is better)	2.7353	2.2647	2.0588	4.1176	3.8235

TABLE III
AVERAGE EXECUTION TIME (IN SECONDS)

DATASETS	COMPOSE(α shape)	COMPOSE (GMM)	FAST COMPOSE	SCARGC(1-NN)	SCARGC(SVM)
1CDT	17.6604 (4)	0.609422 (2)	**0.338512** (1)	18.3254 (5)	3.91082 (3)
1CHT	17.3164 (4)	0.614563 (2)	**0.345122** (1)	18.2382 (5)	3.77782 (3)
1CSurr	59.3483 (4)	34.4966 (3)	**1.69453** (1)	88.998 (5)	24.171 (2)
2CDT	17.7708 (5)	0.647412 (2)	**0.382842** (1)	15.2904 (4)	4.65729 (3)
2CHT	18.6185 (5)	0.69017 (2)	**0.407806** (1)	11.6092 (4)	3.15182 (3)
4CE1CF	190.361 (3)	6.50829 (2)	**2.74359** (1)	309.128 (5)	207.594 (4)
4CR	153.289 (5)	7.98227 (2)	**3.18295** (1)	140.142 (4)	83.2557 (3)
4CRE-V1	141.024 (4)	6.27852 (2)	**1.61991** (1)	286.963 (5)	21.9151 (3)
4CRE-V2	195.801 (4)	5.2888 (2)	**2.60481** (1)	421.596 (5)	56.5427 (3)
5CVT	26.5559 (4)	0.905739 (2)	**0.572302** (1)	52.978 (5)	7.96208 (3)
FG_2C_2D	213.067 (4)	25.1987 (2)	**1.83148** (1)	timeout (5)	73.7001 (3)
GEARS_2C_2D	211.246 (4)	85.2689 (3)	**3.1497** (1)	timeout (5)	65.011 (2)
MG_2C_2D	214.771 (4)	11.4362 (2)	**2.28884** (1)	timeout (5)	71.8093 (3)
UG_2C_2D	106.335 (4)	1.06045 (2)	**0.745** (1)	447.573 (5)	41.0892 (3)
UG_2C_3D	timeout (4.5)	2.23306 (2)	**1.68046** (1)	timeout (4.5)	84.5405 (3)
UG_2C_5D	timeout (4.5)	2.22986 (2)	**1.71864** (1)	timeout (4.5)	384.999 (3)
keystroke	timeout (5)	2.9876 (4)	1.44763 (3)	0.982431 (2)	**0.632132** (1)
Average Rank (lower is better)	4.2353	2.2353	1.1176	4.5882	2.8235

TABLE IV
STATISTICAL SIGNIFICANCE AT $p \leq 0.05$ FOR CLASSIFIER ACCURACY

	COMPOSE(α shape)	COMPOSE (GMM)	FAST COMPOSE	SCARGC(1-NN)	SCARGC(SVM)
COMPOSE(α shape)	n/a				
COMPOSE (GMM)		n/a		$\leftarrow$	$\leftarrow$
FAST COMPOSE			n/a	$\leftarrow$	$\leftarrow$
SCARGC(1-NN)		$\uparrow$	$\uparrow$	n/a	
SCARGC(SVM)		$\uparrow$	$\uparrow$		n/a

TABLE V
STATISTICAL SIGNIFICANCE AT $p \leq 0.05$ FOR CLASSIFIER EXECUTION TIME

	COMPOSE(α shape)	COMPOSE (GMM)	FAST COMPOSE	SCARGC(1-NN)	SCARGC(SVM)
COMPOSE(α shape)	n/a	$\uparrow$	$\uparrow$		
COMPOSE (GMM)	$\leftarrow$	n/a		$\leftarrow$	
FAST COMPOSE	$\leftarrow$		n/a	$\leftarrow$	$\leftarrow$
SCARGC(1-NN)		$\uparrow$	$\uparrow$	n/a	$\uparrow$
SCARGC(SVM)			$\uparrow$	$\leftarrow$	n/a

REFERENCES

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

[2] J. Kolter and M. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *Journal of Machine Learning Research*, vol. 8, pp. 2755–2790, July 2007.

[3] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset shift in machine learning*. The MIT Press, 2009.

[4] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of statistical planning and inference*, vol. 90, no. 2, pp. 227–244, 2000.

[5] G. Ditzler and R. Polikar, "Incremental learning of concept drift from streaming imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, pp. 2283–2301, 2013.

[6] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," *ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 377–382, 2001.

[7] G. Ditzler and R. Polikar, "Semi-supervised learning in nonstationary environments," *International Joint Conference on Neural Networks*, pp. 2741–2748, 2011.

[8] K. D. Robert Capo and R. Polikar, "Active learning in nonstationary environments," *International Joint Conference on Neural Networks*, pp. 1–8, 2013.

[9] G. Marrs, R. Hickey, and M. Black, "The impact of latency on online classication learning with concept drift," *Knowledge Science, Engineering and Management*, vol. 6291.

[10] K. B. Dyer, R. Capo, and R. Polikar, "Compose: A semisupervised learning framework for initially labeled nonstationary streaming data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 12–26, 2014.

[11] G. Krempl, "The algorithm apt to classify in concurrence of latency and drift," *Intelligent Data Analysis*, pp. 223–233, 2011.

[12] V. M. A. Souza, D. F. Silva, J. Gama, and G. E. A. P. A. Batista, "Data stream classication guided by clustering on nonstationary envi- ronments and extreme verication latency," *SIAM International Conference on Data Mining*, pp. 873–881, 2015.

[13] V. M. A. Souza, D. F. Silva, G. E. A. P. A. Batista, and J. Gama, "Classification of evolving data streams with infinitely delayed labels," *IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 214–219, 2015.

[14] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," *Very Large Data Bases*, vol. 29, pp. 81–92, 2003.

[15] R. Capo, A. Sanchez, and R. Polikar, "Core support extraction for learning from initially labeled nonstationary environments using compose," *International Joint Conference on Neural Networks*, 2014.

[16] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," Carnegie Mellon University, Tech. Rep. Technical Report CMU-CALD-02-107, 2002.

[17] K. Bennett and A. Demiriz, "Semi-supervised support vector machines," *Advances in Neural Information processing systems*, pp. 368–384, 2002.

[18] J. Sarnelle, A. Sanchez, R. Capo, J. Haas, and R. Polikar, "Quantifying the limited and gradual concept drift assumption," *International Joint Conference on Neural Networks*, 2015.

[19] V. M. A. Souza, D. F. Silva, J. Gama, and G. E. A. P. A. Batista, "Nonstationary environments - archive," https://sites.google.com/site/nonstationaryarchive/, 2015, [Accessed: 1- Aug- 2016].