This full text paper was peer-reviewed at the direction of IEEE Instrumentation and Measurement Society prior to the acceptance and publication.

Adding Adaptive Intelligence to Sensor Systems with MASS

Christopher Frederickson, Thomas Gracie III, Steven Portley, Michael Moore, Daniel Cahall, Robi Polikar
Rowan University
Department of Electrical and Computer Engineering
{fredericc0, graciet8, portleys4, moorem6, cahalld0}@students.rowan.edu, polikar@rowan.edu

***Abstract*—In sensor systems, tracking gradual drift in a non-stationary environment is a challenging problem. The problem, a phenomenon also known as concept drift, is made even more difficult if the streaming data only consists of unlabeled data after initialization. This scenario is typically referred to as extreme verification latency (EVL), and is common in many sensor applications. In our previous work, we introduced a framework called COMPOSE (COMPacted Object Sample Extraction), which can handle the extreme verification latency problem, provided that the drift is limited. In this paper, we introduce a derivative of COMPOSE called MASS (Modular Adaptive Sensor System) as a solution to extreme verification latency in streaming sensor data, regardless of the particular application. To analyze the performance of MASS, the classification accuracy and execution time were compared to several variations of COMPOSE on synthetic benchmark datasets. The algorithm was then implemented on an Arduino sumo robot, where the objective was to keep the robot within a specific zone based on drifting data returned by the reflectance sensor.**

I. Introduction

With the advent of the Internet of Things (IoT), sensors are becoming more frequent in embedded systems and in many applications from homes and offices to industrial facilities [1]. As sensors become more common and capable of operating in a variety of situations, it is important that these sensor systems are designed to be highly robust and affordable. Due to these advancements, research efforts to increase sensor capabilities have become increasingly important.

Many commonly used sensors exhibit a gradual drift in the readings over time due to long term degradation or changes of the sensor. Sensors are also subject to constantly changing ambient conditions, such as light levels as the day progresses and temperature values as the seasons change. This phenomenon is known as *concept drift*, and it refers to a scenario where the statistical properties of a measured quantity are changing due to either the conditional changes to the output (*real drift*), changes to the distribution of input (*virtual drift*), or both [2]. As a result, sensors need to be recalibrated regularly, which can be a costly and inefficient process. Furthermore, in cases such as sensors on satellites and other extreme environments, external regulation is impossible.

Concept drift is especially common in gas sensors as they are exposed to chemicals which poison and degrade the sensors over their lifetime, which affects the integrity and reliability of the data. Several papers have examined the effects of this drift on E-nose applications, a type of complex gas sensor, and attempted to mitigate sensor drift through compensation. This compensation often relies on complex algorithms to compute predicted drift [3] [4]. Pressure sensors also exhibit drift often dealt with using regular recalibration. The issue of drift is exacerbated in low cost pressure sensors as more as increasing the accuracy and stability of these parts adds cost.

In many applications, after deployment of the sensor system, the developer may have little to no ability to modify or even monitor the performance. These sensors must then be able to adapt to their environment without needing to be adjusted manually. To solve this problem, it can be approached similarly to a problem called *extreme verification latency*. In machine learning literature, verification latency refers to a scenario where labels for training data are not available until sometime after a prediction has been made on the current state of the environment [5]. The length of this delay may not be known beforehand, and may also vary with time. In the case of extreme verification latency, the delay is infinitely long, meaning that no proper labels are available after initialization. Such an environment is referred to as an initially labeled streaming environment (ILSE), and it requires new classification methods capable of adapting to potential changes over time without external regulation.

In this paper, we propose Modular Adaptive Sensor System (MASS), a general approach to extreme verification latency which allows sensors to adapt to drifting concepts independently of the particular sensor application. MASS follows two steps to do so: 1) apply a batched, centroid based clustering algorithm to initially classify the starting state; 2) update the centroids continuously based an update rule. With modularity in mind, MASS was designed such that any clustering algorithm can be used for the initial stage, and any centroid-based clustering algorithm can be used in the update stage. In this way, it functions as not just an algorithm, but a framework which can be modified to fit the needs of any particular sensor application.

II. Related Work

One approach to handle sensor drift was studied by applying a similar algorithm to account for drift in chemical sensors [6]. A set of reference patterns, used to represent the current state of each class, were initialized to the initial state before drift. Every measurement afterwards was then classified by finding

978-1-5090-3202-0/17/$31.00 ©2017 IEEE

the reference pattern that is closest to the measurement a using a geometric discriminant, such as Euclidean distance. These reference patterns were then adjusted after every measurement to become closer to the new measurement. It was shown that this approach proved effective whenever there was an equal representation of each class. When a particular class had not appeared in the data for an extended period of time, it was allowed to drift far enough to where the measurement and the reference pattern were not similar enough for a correct classification.

Compacted Object Sample Extraction, or COMPOSE, is an algorithm designed to track a nonstationary environment in an extreme verification latency setting [7], such as in a sensor system. While the original research did not focus on a specific application, tracking drift in sensor data is an example of an initially labeled, nonstationary streaming environment. The algorithm combines initially labeled data with incoming unlabeled data to train a semi-supervised learning (SSL) algorithm and classify the unlabeled data. Once all the data has been classified, the instances which lie in the center or core region, called core supports, are extracted using a core support extraction (CSE) method such as alpha shape compaction or Gaussian Mixture Models (GMM) [8]. The core supports are then labeled based on the distribution they were derived from. When new unlabeled data arrives, the process is repeated, using the extracted core supports as the labeled data for the SSL. A modification to COMPOSE, called FAST COMPOSE, was implemented later which removed the core support extraction step, and improved the speed of the algorithm with little to no cost to the overall performance [9].

III. Approach

In the initial stage, a standard centroid based classification algorithm is used to cluster an initial set of labeled training data. The algorithm is initialized with a set of M labeled data, L^0, and corresponding labels, Y^0, of classes $1, \ldots, C$ in step 1. A batched clustering algorithm, **CLUSTER**, with relevant free parameters is executed on the initial set of data to determine initial clustering, I^0, and corresponding k cluster centroids, M in step 2. The clustering algorithm can be chosen to fit the particular needs of an application. The resulting cluster centroids, M, are then labeled according to the class that appears most frequently in the cluster in 3. This stage can be performed either directly on the embedded device or offline using a set of training data. In the implementation tested in this paper, the centroid based batched clustering algorithm chosen was k-means clustering using a k-means++ initialization due to the simplicity of implementation on an embedded platform.

In the update stage, the centroid of each cluster is adapted based on the streaming input data. At each timestep, a single data point, x_t, is received in step 5. The data point is given the label of the nearest cluster centroid in step 6. The cluster centroid is then updated according to some update scheme in step 7. Any centroid based clustering algorithm can be used to update the centroids such as online kmeans clustering, inverse weighted kmeans clustering, or k-harmonic means [10]. In this

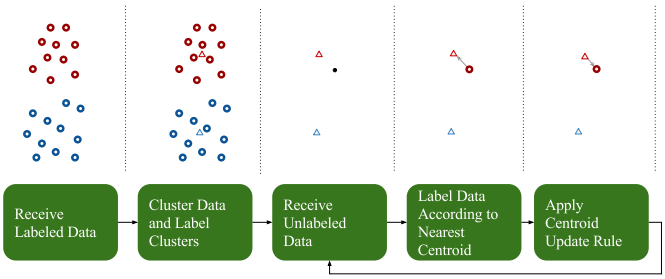

Fig. 1. A diagram of MASS execution. Each colored circle represents a labeled datapoint of a given class. Colored triangles represent the centroid of a cluster of a given class. Black circles indicate new unlabeled data. Steps 3 through 5 are repeated for each new unlabeled datapoint.

implementation, online k-means clustering is used to update the cluster parameters due to the simplicity of execution. The algorithm updates the cluster by first classifying the new datapoint based on the nearest centroid and subsequently shifting the centroid towards the new datapoint based on a learning rate, alpha, as in Equation 1. This simple update scheme allows the algorithm to be efficiently run on any embedded device. The algorithm pseudocode can be found in Algorithm 1. A visual explanation of MASS is shown in Figure 1.

$$\mu_{k,t+1} = \mu_{k,t} + \alpha\left(x_i - \mu_{k,t}\right) \tag{1}$$

Algorithm 1 MASS

Input: Batched Centroid Clustering Algorithm with relevant free parameters - **CLUSTER**; Online Clustering Algorithm Update Rule - **UPDATE**; Learning Rate (α)

1: Receive labeled data
$L^0 = \{x_l \in X$, l = 1,...,M $\}$,
$Y^0 = \{y_l \in Y = \{1, \ldots, C\}, l = 1, \ldots, M\}$
2: Call **CLUSTER** with L^0 and relevant free parameters to obtain the initial clustering
$I^0 = \{i_l \in K = \{1, \ldots, k\}, l = 1, \ldots, M\}$
and cluster centroids $M = \{\mu_i \in X$, i = 1,...,k $\}$
3: Apply a label to each cluster centroid according to the most frequent class
4: **for** $t = 1, 2, \ldots.$ **do**
5: Receive unlabeled data $x_t \in X$
6: Classify data by nearest centroid
7: Apply **UPDATE** with α to update the nearest cluster centroid
8: **end for**

The memory requirements of MASS are significantly reduced as compared to FAST COMPOSE or any batched algorithm. Any algorithm that operates on batches of data must have at least enough memory to store an entire batch whereas MASS must only store the current data point and the cluster centroids. Additionally, many batched algorithms require that data from each class be present in every timestep or the class will be forgotten, necessitating the need for larger batch sizes

with a larger memory footprint. As MASS operates in an online fashion, this is not an issue. Also, in batched algorithms the resulting hypothesis only becomes available after the entire batch is processed, limiting its use in real time applications.

Although MASS provides a simple solution to adapting to concept drift, it requires that certain conditions be met in to perform well. Between any two time-steps or samples of the sensor, the drift must be gradual with no abrupt shifts. If sensor drift causes data to abruptly shift closer to the centroid of a different class, the data will be perpetually mislabeled. All algorithms operating under extreme verification latency have the limitation that the drift must be gradual or limited as there are no true labels to determine what class a given cluster is after an abrupt shift. In situations with abrupt drift, the sampling rate can be increased so that the drift becomes more gradual.

As the only statistic used to identify a cluster is the centroid, the result of a centroid based clustering algorithm must be meaningful. Each combination of clustering algorithms and update schemes will have different limitations and must be chosen for a particular application. Using k-means and the online k-means update scheme, the number of cluster centroids must be known in advance, the variance of each cluster must be approximately the same, and each cluster must be approximately spherical. Using the online k-means update scheme, the algorithm is also sensitive to outliers as they can suddenly shift the cluster centroids.

IV. Experiments and Results

A. Experimental Setup

The algorithm was tested with both a standard benchmark run in an offline fashion and on a robotics platform. The synthetic dataset was provided by the authors of [11] as benchmark for the incremental learning and extreme verification latency community. The benchmark contains 17 datasets with cardinality between 1,600 and 200,000 and number of features between 2 and 10. The first 16 datasets are synthetically generated containing a variety of cluster drifts such as two distributions passing horizontally and two distributions gear shaped rotating. The final dataset, keystroke, was created by logging the time between keystrokes of four individuals as they type a specific phrase. The individual's typing dynamics change as they become faster at typing the phrase, resulting in drift in the dataset. A summary of each dataset is found in Table I. The MASS algorithm was tested against COMPOSE using Alpha Shapes and cluster and label (CNL), COMPOSE using Gaussian Mixture Models (GMM) and CNL, and Fast COMPOSE. All algorithms were executed 500 times on each dataset on a server with 2 x Intel Xeon E5-2670v3 processors and 64 GB DDR4 memory and the results averaged.

The MASS system was implemented on a sumo robot, where the goal was to keep the robot inside of a particular zone. Two different colors of construction paper were used to differentiate the zones. Five different samples of the reflectance of each surface, with labels, were collected and used by MASS as training data. As the sumo robot moved around the ring, each point was classified so that the robot would know if it was approaching the boundary of the ring. Based on this classification, the robot would decide whether or not to turn around. During the classification process, the cluster centroids were modified based on an alpha value of 0.025, as seen in Equation 1. To test the adaptability of MASS, an array of infrared LEDs was constructed. While the sumo robot was running, the LED array intensity was varied. On each run the test environment was reset by returning the LED array to its initial intensity and taking a new set of initialization data.

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

A sensitivity analysis was performed in order to understand the robustness of MASS to changes in learning rate and number of clusters. The learning rate and number of clusters was varied for the Sumo dataset and keystroke dataset and the accuracy plotted.

B. Results

Table II shows the average accuracy and Table IV shows the average execution time (in seconds) of the algorithms. The ranking of each algorithm for each dataset is shown in parenthesis. In order to determine the statistical significance between each algorithm, the Friedman test was applied with the corresponding Nemenyi post-hoc test. No statistically significant difference (at $\alpha = 0.05$) was found between the accuracies of each algorithm. The results of the post-hoc test comparing the statistical significance (at $\alpha = 0.05$) of execution time is found in Table IV. Empty cells indicate that the difference between the execution time of each algorithm was not statistically significant. Either an up arrow ($\uparrow$) or left arrow ($\leftarrow$) indicate that the difference between the two algorithms are statistically significant and the direction of the arrow indicates the faster algorithm. The MASS algorithm had no statistically significant decrease in accuracy compared to COMPOSE despite operating in an incremental fashion and only having access to a single data point at a time. The MASS algorithm was also competitive with COMPOSE with an execution time

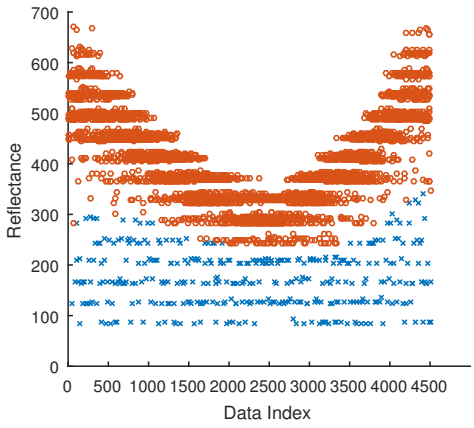

Fig. 2. Classification data from the Sumo robot where the red circles represent a classification hypothesis of the center region and the blue crosses represent the outer border.

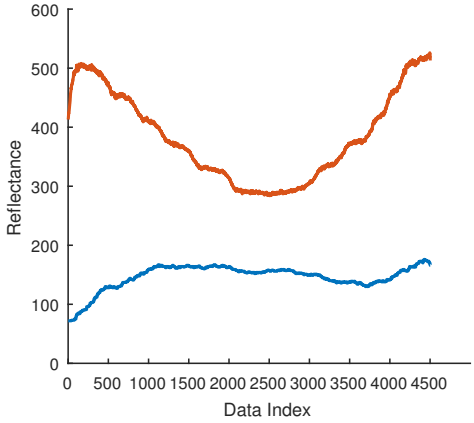

Fig. 3. Cluster mean data from the Sumo robot where each line indicates the mean of the cluster at a given data index.

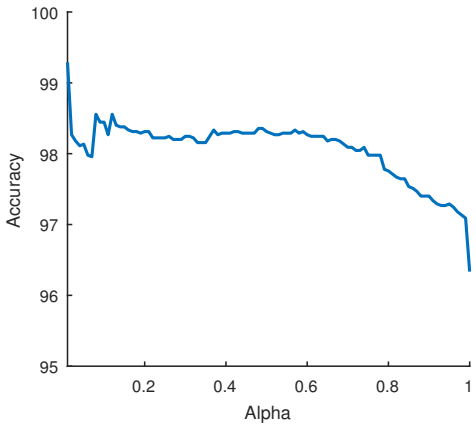

Fig. 4. A sensitivity analysis of alpha on the Sumo dataset run with the number of clusters set to 2. The accuracy of MASS is insensitive to the the choice of alpha.

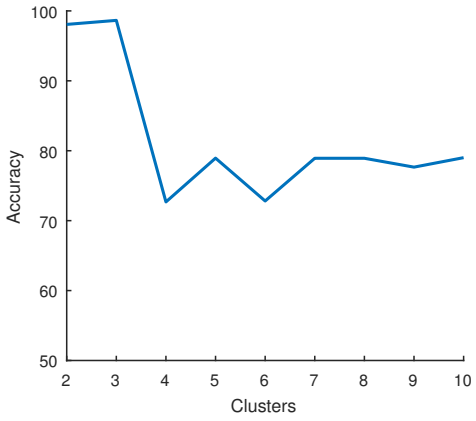

Fig. 5. A sensitivity analysis of the number of clusters on the Sumo dataset run with the learning rate set to 0.05. A small number of clusters is necessary for accurate performance.

on similar hardware being statistically significantly faster than COMPOSE alpha shapes and statistically significantly slower than FAST COMPOSE. Although the execution time of MASS was longer than FAST COMPOSE, by only operating on a single data point at a time, the memory required to operate the algorithm is greatly reduced, a necessity on an embedded platform.

The classifier hypothesis of the Sumo dataset is shown in Figure 2 where the red circles represent the center region and the blue crosses represent the outer region. With the ambient lighting changing, the Sumo robot stayed within the center region and achieved a classifier accuracy of 99.62%. The cluster means at each data index is shown in Figure 3.

In order to test the robustness of MASS, a sensitivity analysis was performed varying the learning rate and number of clusters on the keystroke and Sumo datasets. With the Sumo dataset, MASS was insensitive to the choice of learning rate as in Figure 4, but required a fairly precise number of clusters to perform with high accuracy shown in Figure 5.

With the keystroke dataset, MASS was robust to the number of clusters chosen as in Figure 7, but required a fairly precise choice of learning rate as in Figure 6.

V. Conclusion & Future Work

In this effort, we introduced the online MASS algorithm which allows classification to be performed in an extreme verification latency setting. MASS is designed to be computationally simple, use little memory, and capable of being added into a sensor system with little effort from the system designer. The algorithm can be used to implement adaptive capabilities in low power IoT devices that would otherwise run the risk of misreading data due to drifting concepts. Despite its significantly reduced complexity compared to COMPOSE, the MASS algorithm performed remarkably well, with statistical tie to the performance of the better established COMPOSE variations. We note that the overall goal of this effort was not to design a classifier that can beat COMPOSE, but rather a classifier that can replicate the COMPOSE performance reasonably well, but has the added advantage of being able to run on an embedded system.

Future work includes testing MASS on a more comprehensive dataset to ensure that the algorithm performs accurately

TABLE II
AVERAGE ACCURACY

DATASETS	COMPOSE (α shape)	COMPOSE (GMM)	FAST COMPOSE	MASS
1CDT	99.9615 (3.5)	99.9615 (3.5)	99.9679 (2)	**99.9744** (1)
2CDT	96.5769 (2)	**96.609** (1)	95.1731 (4)	96.359 (3)
1CHT	99.6026 (2)	**99.6154** (1)	99.5705 (3)	99.5513 (4)
2CHT	90.391 (2)	**90.4423** (1)	89.4103 (4)	89.9167 (3)
4CR	99.9924 (2.5)	99.9924 (2.5)	99.9924 (2.5)	99.9924 (2.5)
4CRE-V1	80.7883 (4)	**98.5887** (1)	97.7484 (3)	98.0871 (2)
4CRE-V2	**92.5907** (1)	92.5857 (2)	92.4632 (3)	92.3978 (4)
5CVT	57.9676 (3)	49.9304 (4)	81.3348 (2)	**89.9** (1)
1CSurr	90.9524 (2)	84.6515 (4)	**95.641** (1)	90.6722 (3)
4CE1CF	93.9204 (2)	93.9012 (3)	**93.9528** (1)	93.8104 (4)
UG-2C-2D	95.6347 (2)	**95.6378** (1)	95.6112 (3)	95.4724 (4)
MG-2C-2D	**93.1162** (1)	92.0298 (4)	93.0167 (2)	92.8303 (3)
FG-2C-2D	87.897 (4)	95.5454 (2)	**95.5813** (1)	95.3919 (3)
UG-2C-3D	timeout (4)	**95.198** (1)	95.1202 (2)	94.9758 (3)
UG-2C-5D	timeout (4)	82.9258 (3)	**91.9889** (1)	91.4985 (2)
GEARS-2C-2D	90.9792 (3)	82.2135 (4)	91.2637 (2)	**98.6111** (1)
Keystroke	timeout (4)	87.4921 (1)	85.9204 (2)	85.6383 (3)
Average Rank (lower is better)	2.7059	2.2941	2.2647	2.7353

TABLE III
AVERAGE EXECUTION TIME (IN SECONDS)

DATASETS	COMPOSE (α shape)	COMPOSE (GMM)	FAST COMPOSE	MASS
1CDT	17.6604 (4)	0.609422 (3)	**0.338512** (1)	0.421824 (2)
2CDT	17.7708 (4)	0.647412 (3)	**0.382842** (1)	0.410426 (2)
1CHT	17.3164 (4)	0.614563 (3)	**0.345122** (1)	0.458375 (2)
2CHT	18.6185 (4)	0.69017 (3)	**0.407806** (1)	0.465429 (2)
4CR	153.289 (4)	7.98227 (3)	**3.18295** (1)	3.43345 (2)
4CRE-V1	141.024 (4)	6.27852 (3)	**1.61991** (1)	2.97164 (2)
4CRE-V2	195.801 (4)	5.2888 (3)	**2.60481** (1)	4.27871 (2)
5CVT	26.5559 (4)	0.905739 (2)	**0.572302** (1)	1.07329 (3)
1CSurr	59.3483 (4)	34.4966 (3)	**1.69453** (1)	1.92582 (2)
4CE1CF	190.361 (4)	6.50829 (3)	**2.74359** (1)	4.52267 (2)
UG-2C-2D	106.335 (4)	1.06045 (2)	**0.745** (1)	1.9527 (3)
MG-2C-2D	214.771 (4)	11.4362 (3)	**2.28884** (1)	4.92792 (2)
FG-2C-2D	213.067 (4)	25.1987 (3)	**1.83148** (1)	5.0532 (2)
UG-2C-3D	timeout (4)	2.23306 (2)	**1.68046** (1)	3.82042 (3)
UG-2C-5D	timeout (4)	2.22986 (2)	**1.71864** (1)	3.87683 (3)
GEARS-2C-2D	211.246 (4)	85.2689 (3)	**3.1497** (1)	9.1568 (2)
Keystroke	timeout (4)	2.9876 (3)	**1.44763** (1)	1.46271 (2)
Average Rank (lower is better)	4	2.7647	1	2.2353

TABLE IV
STATISTICAL SIGNIFICANCE AT $\alpha = 0.05$ FOR CLASSIFIER EXECUTION TIME

	COMPOSE (α shape)	COMPOSE (GMM)	FAST COMPOSE	MASS
COMPOSE (α shape)	n/a	$\uparrow$	$\uparrow$	$\uparrow$
COMPOSE (GMM)	$\leftarrow$	n/a	$\uparrow$	
FAST COMPOSE	$\leftarrow$	$\leftarrow$	n/a	$\leftarrow$
MASS	$\leftarrow$		$\uparrow$	n/a

in a wider variety of scenarios. Additionally, alternate batch and online clustering algorithms should be tested in the MASS framework to better understand the implications of the choice in clustering algorithms.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under grant no. 1310496 and grant no. 1429467.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 44, 2014.

[3] F. A. Davide, C. Di Natale, and A. D'Amico, "Self-organizing multisensor systems for odour classification: internal categorization, adaptation and drift rejection," *Sensors and Actuators B: Chemical*, vol. 18, no. 1-3, pp. 244–258, 1994.

[4] D. Hui, L. Jun-Hua, and S. Zhong-Ru, "Drift reduction of gas sensor by wavelet and principal component analysis," *Sensors and Actuators, B: Chemical*, vol. 96, no. 1-2, pp. 354–363, 2003.

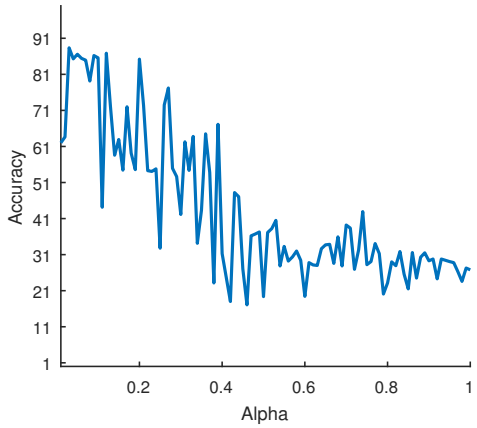

Fig. 6. A sensitivity analysis of alpha on the keystroke dataset with the number of clusters set to 20.

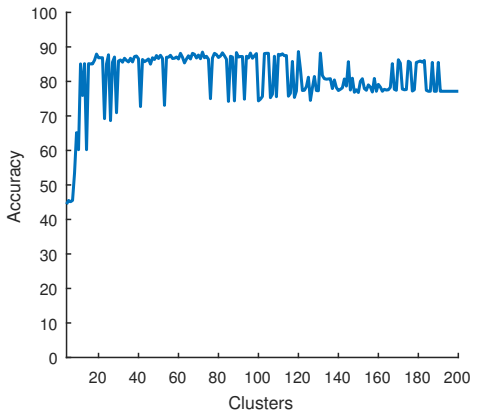

Fig. 7. A sensitivity analysis of the number of clusters on the keystroke dataset with the learning rate set to 0.05. The classifier is relatively insensitive to the number of clusters after a

[5] G. R. Marrs, R. J. Hickey, and M. M. Black, "The impact of latency on online classification learning with concept drift," in *International Conference on Knowledge Science, Engineering and Management*. Springer, 2010, pp. 459–469.

[6] M. Holmberg, F. Winquist, I. Lundström, F. Davide, C. DiNatale, and A. D'Amico, "Drift counteraction for an electronic nose," *Sensors and Actuators B: Chemical*, vol. 36, no. 1, pp. 528–535, 1996.

[7] 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.

[8] R. Capo, A. Sanchez, and R. Polikar, "Core Support Extraction for Learning from Initially Labeled Nonstationary Environments using COMPOSE," 2014.

[9] M. Umer, C. Frederickson, and R. Polikar, "Learning under extreme verification latency quickly: Fast compose," in *IEEE Symposium Series on Computational Intelligence 2016*. IEEE.

[10] W. Barbakh and C. Fyfe, "Online clustering algorithms," *International Journal of Neural Systems*, vol. 18, no. 03, pp. 185–194, 2008.

[11] V. M. A. Souza, D. F. Silva, J. Gama, and G. E. A. P. A. Batista, "Data stream classification guided by clustering on nonstationary environments and extreme verification latency," in *Proceedings of SIAM International Conference on Data Mining (SDM)*, 2015, pp. 873–881.