Data stream classification guided by clustering on nonstationary environments and extreme verification latency

Data Stream Classification Guided by Clustering on Nonstationary Environments and Extreme Verification Latency

Vinícius M. A. Souza* Diego F. Silva* João Gama† Gustavo E. A. P. A. Batista*

Abstract

Data stream classification algorithms for nonstationary environments frequently assume the availability of class labels, instantly or with some lag after the classification. However, certain applications, mainly those related to sensors and robotics, involve high costs to obtain new labels during the classification phase. Such a scenario in which the actual labels of processed data are never available is called extreme verification latency. Extreme verification latency requires new classification methods capable of adapting to possible changes over time without external supervision. This paper presents a fast, simple, intuitive and accurate algorithm to classify nonstationary data streams in an extreme verification latency scenario, namely *Stream Classification Algorithm Guided by Clustering* – SCARGC. Our method consists of a clustering followed by a classification step applied repeatedly in a closed loop fashion. We show in several classification tasks evaluated in synthetic and real data that our method is faster and more accurate than the state-of-the-art.

Keywords: Data stream classification, concept drift, extreme verification latency.

1 Introduction

Most of the research on data stream classification in the presence of concept drifts assumes the availability of class labels, instantly or with some lag after the classification, for example [3] and [7]. In summary, these methods verify if the current model is outdated based on classification performance information obtained on the most recent labeled data.

This assumption is perfectly feasible for a number of applications. For instance, consider the task of predicting if a stock market share price will increase or decrease in the next hour. We can obtain the actual labels with exactly one hour lag. Another example is the prediction of energy consumption. In particular, it seems that classification tasks derived from regression problems easily fulfill the availability requirements of actual labels after classification.

However, there exists an emerging class of applications mainly in sensors and robotics in which the labeled data availability is not fulfilled. These applications typically involve high costs for labeling data, possibly manually with the assistance of an expert. A recent work [10] pointed these assumptions on the timing and availability of information as one of the main challenges on the current data stream mining research.

The time between a classification and the availability of correspondent actual label is denoted as *verification latency* [11]. When this time approaches infinity, we have an *extreme verification latency* scenario where no labeled data are received after the initialization of the classification model. This scenario differs from the data stream clustering setting in two main ways:

- It is an intrinsically classification task in the sense that the classes are known and we are interested in discriminating the classes apart. In clustering, the clusters are unknown *a priori* and the interpretation of each cluster is given *a posteriori*;
- It assumes the availability of a small amount of initial labeled data. These data are made available *before* the classification starts. Differently from clustering in which no labeled data are available.

The initial labeled data are necessary to define the problem as classification, including the number of classes and their initial disposition in the feature space. Without such data it would be difficult, if not impossible, to automatically interpret the future occurrences of the examples without human intervention. However, in a short period of time these data are outdated and can degrade the current model.

The semi-supervised stream learning is also different from the scenario considered in this work. In semi-supervised learning, the data stream is typically divided into equal-sized chunks where only a small fraction (e.g. 5%) of data is labeled and the goal is to label the remaining unlabeled data in the chunk. Thus, these algorithms can use labeled and unlabeled data together to perform the classification. However, we address here a problem in which there is the need to immediately assign a label to each individual example that arrives. Finally, the scenario of extreme verification latency differs from the active learning setting, in which the algorithm selects a small portion of data to be externally labeled by an oracle and to build an accurate classifier.

*Institute of Mathematics and Computer Science, University of São Paulo, Brazil – {vsouza, diegofsilva, gbatista}@icmc.usp.br

†Faculty of Economics, University of Porto, Portugal – jgama@fep.up.pt

Thus, both semi-supervised and active learning approaches can make use of partially labeled data to monitor the drift rates and update the current classification model whether convenient. In extreme verification latency this is not possible due to the complete lack of labeled data in the classification phase.

For a concrete application that requires extreme verification latency in nonstationary environments, consider a behavioral biometric application where users are recognized by their typing profile as presented in [2]. In this security system, all users type the same password and should be recognized by their typing pace. Such a characteristic evolves over time, so the system need to constantly adapt to the new behavior of each user without any supervision. This application is better approached later in this paper.

We can also find several applications in sensors and robotics. For instance, an autonomous robot can be trained in a specific environment and sent to explore an unknown area without any external aid or human supervision. This area may have different characteristics from those known in the initial model and the robot must adapt to changes without the availability of the actual labels. In sensor applications where the classification task is dependent of environmental measurements, the climate changes can easily degrade a static model and the knowledge of actual label can be involve high costs. An example is the laser sensor that recognizes insect species, as described by [13].

This paper presents a fast, simple, intuitive and accurate algorithm to classify nonstationary data streams in extreme verification latency scenario, namely ***S**tream **C**lassification **A**lgo**r**ithm **G**uided by **C**lustering* – **SCARGC**. Our method consists of a clustering followed by a classification step applied repeatedly in a closed loop fashion. The algorithm uses the current and past cluster positions obtained by clustering the unlabeled data to track the drifting classes over time. We show in 16 synthetic and 2 real classification tasks that our method is faster and more accurate than the state-of-the-art. Due to the importance of evaluating these methods in a concept drift environment with controlled characteristics, this paper also proposes a set of nonstationary synthetic benchmark datasets, and make it publicly available for the data stream community.

2 Related Work

According to the best of our knowledge, the literature has two algorithms that deal with data stream classification in extreme verification latency in nonstationary environments: Arbitrary Sub-Populations Tracker – APT [9] and Compacted Object Sample Extraction – COMPOSE [6].

The APT algorithm considers that each class can be represented as a mixture of arbitrarily distributed sub-populations. Briefly, the algorithm works in a two-stage learning strategy. First, it uses expectation maximization to determine the optimal one-to-one assignment between the unlabeled and labeled data. Next, it updates the classifier to reflect the population parameters of newly received data and the drift parameters, relating the previous time step to the current one. This algorithm has some drawbacks. For instance, it has the assumption that the number of instances of each subpopulation is constant along the time. This limitation is hard to be fulfilled and may degrade the method after some iterations. Furthermore, the tests performed in [6] indicate a considerable high computational cost.

The COMPOSE framework follows three steps: i) it combines initial labels with new unlabeled data to train a semi-supervised classifier and label the current unlabeled data; ii) for each class, it constructs α-shapes (a generalization of convex hull), providing a tight envelope around the data that represent the current class conditional distribution; and iii) it compacts the α-shape and extracts representative instances (called core supports) from the compacted α-shape, which now represent the geometric center of each class distribution. The process is repeated iteratively when new unlabeled data arrive, where the core supports from the previous iterations are now considered labeled instances for the current iteration.

The main weakness of COMPOSE is the α-shape construction method that is a computationally expensive process on high dimensional data. Furthermore, the algorithm has two important parameters: α and CP. The α parameter is related to the level of detail to model shapes of classes using the convex hull algorithm. For a large value of α the resultant shape is the convex hull of the points and for a small value of α the resultant shape may become concave or even contain disconnected regions. The CP parameter is related to the level of compaction applied to the α-shapes. As pointed by the authors, both parameters affect the performance of COMPOSE, mainly the choice of values for the CP parameter. If α-shapes are too compact, relevant instances of the future distribution are lost. In contrast, if the α-shapes are little compacted, instances of rival classes may overlap.

3 Proposed Method

In this paper, we propose an algorithm to classify data streams in nonstationary environments with extreme verification latency. We consider the scenario where the actual labels of unlabeled data are never available as a guidance to update the classification model over time.

We named our method ***S**tream **C**lassification **A**lgo**r**ithm **G**uided by **C**lustering* – **SCARGC**. The SCARGC is intuitive and has a low implementation complexity. Its simplicity has a direct impact on computational complexity without affecting the accuracy performance as discussed in Section 4.

We begin by stating and justifying the main assumptions of our approach:

1. An initial reduced set of labeled data is available. Such data are necessary to define the problem as classification, including the number of classes and the disposition of the classes in the feature space;

2. The concepts drift incrementally. Incremental drift is a necessary condition to allow the classes to be tracked without labels. Many application domains lead naturally to incremental drift. One example are drifts caused by environment changes (e.g. temperature) that occur continuously over time. This type of drift is often considered more challenging to detect than abrupt, given the significant overlap between concepts in a short period of time [6];

3. The number of classes is known and constant over time. Although some applications have a variable or unknown number of classes, frequently we are primarily interested in one of these classes, thus the problem can be transformed into a binary classification problem.

Assumptions 1 and 2 are inherent to the problem and are present in all other approaches proposed in the literature. Assumption 3 is inherent to our solution and will be the focus of future work. Although these assumptions may appear restrictive in a first glance, we believe the standard assumption of immediate availability of actual class labels is even more restrictive, and even impossible to fulfill in many application domains.

Given our assumptions, we are now in position to describe our approach. We start building an initial classification model using the available labeled data with c classes. We also divide the initial labeled data into $k \geq c$ clusters. If $k = c$, we simply use the c classes as initial clusters. If $k > c$, we run a clustering algorithm, and associate each cluster to one class, as we describe later. In both cases, we denote the initial set of k clusters by $\mathcal{C}^{(0)} = \{\mathcal{C}_1^{(0)}, \mathcal{C}_2^{(0)}, \ldots, \mathcal{C}_k^{(0)}\}$.

As the classification phase begins and the algorithm starts receiving new unlabeled data, two actions are immediately taken: each example is stored in a pool and has its label predicted by the initial classification model. After a minimal amount of examples of each class is stored in the pool, the pool of examples is clustered into k clusters. We denote the new set of clusters as $\mathcal{C}^{(1)} = \{\mathcal{C}_1^{(1)}, \mathcal{C}_2^{(1)}, \ldots, \mathcal{C}_k^{(1)}\}$. At this point, we associate each cluster $\mathcal{C}_i^{(1)} \in \mathcal{C}^{(1)}$ with one cluster $\mathcal{C}_j^{(0)} \in \mathcal{C}^{(0)}$. Consequently, each cluster $\mathcal{C}_i^{(1)}$ is assigned to one class. Such association allows us to label each example in the pool. The objective is to map the previous concept to the new concept found in the clustering phase.

The recently labeled examples are used to create an updated classification model that replaces the initial one. The algorithm goes in a closed loop, alternating clustering and classification. The classification model is build over the past labeled data, with labels given by the clustering algorithm. The labels are given by the association of the clusters in the current iteration $\mathcal{C}^t$ with the labels in the previous iteration $\mathcal{C}^{t-1}$.

In this general framework we omitted some details that are parameters or implementation decisions. In our implementation, we favored simplicity and efficiency, since stream mining may require very fast response times. In the experiments performed in this paper, we chose the k-means algorithm as clustering method. The k-means is a simple and computationally efficient algorithm. In addition, the algorithm has many extensions such as X-means [12] and Fuzzy c-means [5] that can be adapted to be used in our framework.

We store the centroids of the past clustering iteration and use it in two ways. First, we use them as seeds for the current clustering step. Such procedure avoids the instability of k-means due to its stochastic nature. Second, we use the centroids as summary statistic about the clusters. We use a simple centroid similarity calculation between the current and previous algorithm iteration using Euclidean distance to perform the mapping between the clusters. Thus, given the current centroids $(q_1, q_2, \ldots, q_k)$ from the most recent unlabeled clusters $\mathcal{C}^t$ and the past centroids $(p_1, p_2, \ldots, p_k)$ from the previously labeled clusters $\mathcal{C}^{t-1}$, where q_i and p_i are n-dimensional data, each centroid p_i have a label y_i and each centroid q_i needs a label $\hat{y}_i$. This label is obtained by the simple nearest neighbor algorithm. For this, each new centroid q_i is associated to its closest past centroid, according to the Euclidean distance. In other words, after calculating the distance of a centroid q_i to each past centroid, the label $\hat{y}_i$ given to q_i is the same of the label y_i of the nearest past centroid.

In our experiments with synthetic data, we start a new clustering step whenever the pool achieves 300 examples and a minimal amount of data points are associated for each class by the classifier (e.g. 10 data points). The pool size is directly related to how often the algorithm can adapt to concept drifts. Although small pool sizes allow fast adaptation, it also

increases computational costs due to the exhaustive verifications of drifts. Small sample sizes may also lead to unrepresentative clusters due to lack of data. On the other hand, a large sample size may contain more than one concept. The influence of variation of pool size is discussed in Section 4.5.

In order to reduce computational costs we do not update the classification model every iteration. Instead, we test whether there is significant difference between the labels given by the current classifier and the clustering algorithm. A significant difference is considered as an indicative of concept drift and a new classification model is built. We reinforce that the purpose of the test is not be an explicit drift detector, but only an indicator that aims to reduce the cost of constant updates. We suggest the use of *Kappa coefficient* [4] as a significance test. However, our framework allows the use of other statistical tests to compare the labels or the data distributions given by the clustering and the current classifier.

Our framework also supports any classification algorithm. In our experiments, we evaluate the behavior of SCARGC with two well known algorithms: 1-Nearest Neighbor (1NN) and Support Vector Machine (SVM).

A more formal description is shown in Algorithm 1, which requires as input an initial set of labeled training examples $\mathcal{T}$ and an unlabeled data stream $\mathcal{DS}$. The algorithm has two parameters: the number of clusters k and the pool size θ.

Algorithm 1: Stream Classification Algorithm Guided by Clustering - SCARGC

Input: Initial training data $\mathcal{T}$; Data Stream $\mathcal{DS}$; Max pool size θ; Number of clusters k
Output: Updated classifier Φ; Label y for each $x \in \mathcal{DS}$

$\Phi \leftarrow buildClassifier(\mathcal{T})$
if $k = c$ **then**
  $\mathcal{C} \leftarrow actual\_classes(\mathcal{T})$
else
  $\mathcal{C} \leftarrow clustering(\mathcal{T}, k)$
$pool \leftarrow \varnothing$
$labels \leftarrow \varnothing$
while $\mathcal{DS}$ *has events* **do**
  $x \leftarrow next\_event(\mathcal{DS})$
  $y \leftarrow \Phi(x)$
  $pool \leftarrow pool \bigcup \{x\}$
  $labels \leftarrow labels \bigcup \{y\}$
  if $|pool| = \theta$ **then**
    if $countExamplesPerClass(labels, c) \geqslant 10$ **then**
      $\mathcal{C} \leftarrow clustering(pool, k)$
      $M \leftarrow matching(\mathcal{C}, labels)$
      if $concordance(M) \neq 1$ **then**
        $\mathcal{T}' \leftarrow label\_data(pool, \mathcal{C})$
        $\Phi \leftarrow buildClassifier(\mathcal{T}')$
        $pool \leftarrow \varnothing$
        $labels \leftarrow \varnothing$

4 Experimental Evaluation

We evaluate SCARGC with synthetic and real data. Synthetic data are important to understand and evaluate the behavior of the algorithm under certain drift conditions that are likely to occur in real data.

In order to promote the reproducibility of our results, we created a website [1] in which we made available all data, source code and some supplemental material was not included in the paper due to space restrictions.

We would like to compare our method with COMPOSE and APT for all synthetic and real datasets. However, the source code for these methods is unavailable and our attempts to reproduce them with our own implementation were unsuccessful. Therefore, we directly compare our method to the other only for the datasets that the authors published detailed experimental results. Such datasets are UG-2C-2D, UG-2C-3D, MG-2C-2D and NOAA. The results from other methods we reproduce here were obtained by estimating the accuracy in plots by eye. Although, we were extremely cautious performing such procedure, we note it has limited accuracy.

4.1 Synthetic Data This paper provides to the stream mining community a set of nonstationary benchmark datasets. More specifically, we provide 16 datasets with incremental changes over time. Table 1 presents a brief description of the datasets[1]. The column *Drift* indicates the interval in number of examples between consecutive changes. The length of datasets varies from 16,000 to 200,000, and amount of classes from 2 to 5.

Table 1: Synthetic datasets description.

Dataset	#Classes	#Feat.	Drift	Length
1CDT	2	2	400	16,000
2CDT	2	2	400	16,000
1CHT	2	2	400	16,000
2CHT	2	2	400	16,000
4CR	4	2	400	144,400
4CRE-V1	4	2	1,000	125,000
4CRE-V2	4	2	1,000	183,000
5CVT	5	2	1,000	40,000
1CSurr	2	2	≈600	55,283
4CE1CF	5	2	750	173,250
FG-2C-2D	2	2	2,000	200,000
UG-2C-2D	2	2	1,000	100,000
UG-2C-3D	2	3	2,000	200,000
UG-2C-5D	2	5	2,000	200,000
MG-2C-2D	2	2	2,000	200,000
GEARS-2C-2D	2	2	2,000	200,000

The datasets UG-2C-2D, UG-2C-3D, and MG-2C-2D were originally proposed in [6]. The datasets UG-

[1] More details about the names, movements performed by the classes over time (in video descriptions) or other characteristics of the data can be seen in the paper website [1].

2C-5D and GEARS-2C-2D were kindly provided by the authors of the same paper, although they do not use them in their evaluations.

4.2 Analysis of Average Accuracy We compare our method against three bounds that simulate the traditional static supervised learning classifier and classifiers that are constantly updated by an oracle:

1. **Static.** The classifier is trained with the first examples from the stream and it is not updated over time;
2. **Sliding.** The classifier is constantly updated whenever a test example is processed. Then, the oldest example is dropped off from the window and the most recent processed example is added with its *actual* label. This setting represents a zero verification latency scenario;
3. **Landmark.** The classifier uses an incremental window that grows with each new test example and their *actual* label. The window starts with N examples and grows until accumulate $2 * N$ examples. When the limit is reached, the N oldest examples are dropped off from the window. This process is repeated during all test phase. The *Landmark* setting also represents a zero verification latency. However, it provides longer lifetime for the training examples compared to the *Sliding* setting.

We included a comparison to *Sliding* and *Landmark*, because they have access to all actual class labels available on each dataset. In term of performance, we can expect that these methods subsume all other methods that have access to partial information such as in semi-supervised or active learning approaches.

Table 2 presents the average accuracy over the entire stream achieved by SCARGC with 1NN and SVM as base classifiers. Due to the variation in the scores, an additional ranking average (Friedman Test) is also provided. The competitor methods *Static, Sliding* and *Landmark* use a 1NN classifier. However, we emphasize that the results with SVM are similar. SCARGC was initially trained with just 50 labeled examples, considerably less than the other approaches that started with 300 examples. We chose such a reduced amount of initial labeled data to stress that our approach can provide competitive results even with a small number of initial instances, which can be expensive to obtain in some application domains. An analysis of variation of amount of initial labeled data is presented in Section 4.5. The pool size of SCARGC is fixed in 300 examples.

Among the proposed bounds, we can note that the *Sliding* setting presents the best performance. Although its performance is very similar to the *Landmark* setting, the *Sliding* has the advantage of using a smaller amount of memory. Thus, throughout the text we will refer only to the *Sliding* setting for comparisons.

Table 2: Average accuracy over the entire stream for synthetic benchmark datasets. The best result for each dataset is in bold.

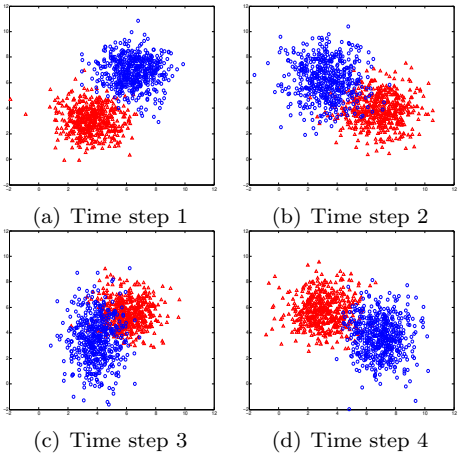
Dataset	Static	Sliding	Land mark	SCARGC (1NN)	SCARGC (SVM)
1CDT	99.07	**99.87**	**99.87**	99.75	99.51
2CDT	55.09	**93.47**	92.29	90.92	90.72
1CHT	94.53	99.23	99.20	**99.25**	99.00
2CHT	54.57	85.43	84.92	**86.02**	85.82
4CR	25.26	**99.98**	**99.98**	99.95	98.48
4CRE-V1	25.22	**97.64**	97.43	97.39	97.48
4CRE-V2	26.26	89.37	89.38	91.90	**91.95**
5CVT	48.12	89.17	86.27	90.15	**90.32**
1CSurr	65.55	**98.51**	98.46	94.35	94.97
4CE1CF	94.70	97.14	**97.21**	94.08	90.29
FG-2C-2D	81.29	93.84	93.89	95.16	**95.23**
UG-2C-2D	45.87	94.26	94.26	**95.56**	95.53
UG-2C-3D	64.09	92.86	92.92	94.77	**94.79**
UG-2C-5D	69.20	89.90	90.08	**90.98**	88.24
MG-2C-2D	51.58	90.40	90.45	92.71	**92.75**
GEARS-2C-2D	94.96	**99.86**	99.85	95.89	95.26
Overall avg.	62.21	**94.43**	94.15	94.30	93.77
Rank avg.	4.875	2.469	2.594	**2.375**	2.688

SCARGC performs very similarly to the *Sliding* for all datasets even without the knowledge of actual labels over time. For more than half of datasets the results are slightly better, as in the case of 1CHT, 2CHT, 4CRE-V2, 5CVT, FG-2C-2D, UG-2C-2D, UG-2C-3D, UG-2C-5D, and MG-2C-2D. These results may sound counter-intuitive in a first sight, since *Sliding* setting have access to the actual labels during the stream. However, such performance differences are due to the effect of overfitting in *Sliding* classifier. The differences are more evident when the classes get closer together and there is a significant performance loss. In these moments, the classifiers that have access to the actual labels are more prone to overfitting due to the large class overlapping. In contrast, the clustering tends to create a smoother class separation borders that are less likely to lead to overfitting.

In some datasets such as 1CDT, 1CHT, 4CE1CF, and GEARS-2C-2D, the separating margin of classes stays static over time. For this reason, the *Static* is able to present very good results even in the presence of feature drifts.

The proposed algorithm achieves its performance using much less labeled data than *Static*, *Landmark* and *Sliding* approaches. Just for comparison purposes, for the largest datasets with 200,000 examples, the initial labeled data represents only 0.02% of all data. Even for smaller datasets, with 16,000 examples, the initial labeled data represents less than 0.5% of all data.

4.3 Unimodal Gaussian Data This section provides detailed experimental results for the unimodal Gaussian dataset UG-2C-2D. In this dataset two bidimensional Gaussian clusters rotate around a common axis. As they rotate, the distance between the Gaussian components varies, making the classes overlap to change with time. Fig. 1 illustrates this dataset in four different moments and Fig. 2 shows the classification performance results over time.



(a) Time step 1 (b) Time step 2

(c) Time step 3 (d) Time step 4

Figure 1: Snapshots over time of UG-2C-2D dataset.

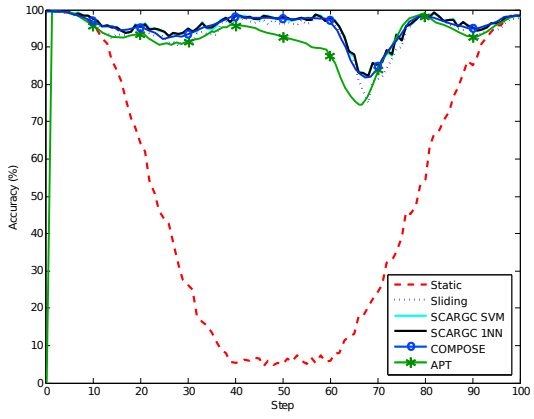
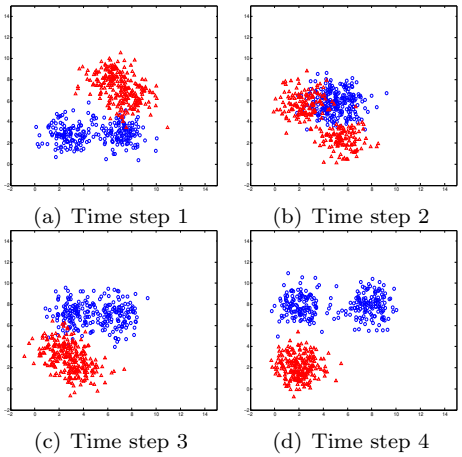


Figure 2: Mean accuracy over time for bidimensional unimodal dataset UG-2C-2D. SCARGC with $k = 2$ and COMPOSE with $CP = 0.70$ and $\alpha = 0.4$.

In order to standardize the presentation of the results, independently of the amount of examples in each dataset, we divided the data in 100 equal batches (when possible) and calculated the mean accuracy achieved in each batch. Although the results are presented as if they were processed into 100 steps, we emphasize that the classifier returns online predictions for each unlabeled example from the stream. In Fig. 2 we can see that SCARGC outperforms APT by a large margin and shows the same performance as COMPOSE during the entire stream. However, SCARGC requires much less computational effort.

4.4 Multimodal Gaussian Data For the multimodal experiment, we show detailed performance results on MG-2C-2D dataset. In Fig. 3, we present 4 snapshots to illustrate the evolving behavior of this data. In Fig. 4 we can see that SCARGC outperforms the *Sliding* classifier, as well as APT and COMPOSE.



(a) Time step 1 (b) Time step 2

(c) Time step 3 (d) Time step 4

Figure 3: Snapshots over time of MG-2C-2D dataset.

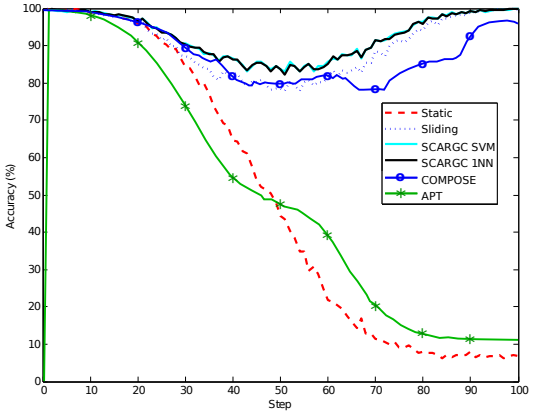


Figure 4: Mean accuracy over time for multimodal dataset MG-2C-2D. SCARGC with $k = 4$. COMPOSE with $CP = 0.70$ and $\alpha = 0.43$.

To achieve good results in MG-2C-2D dataset, it was necessary to use more than one cluster for each class as input for the algorithm. Thus, the results presented here were obtained with the use of four clusters.

4.5 Parameter Sensitivity Analysis SCARGC has three input parameters: *i*) initial labeled data ($\mathcal{T}$); *ii*) pool size (θ); and *iii*) number of clusters (k).

We present here the analysis for initial labeled data size and pool size for 4CRE-V2 and MG-2C-2D datasets. We selected these datasets because they have different characteristics and drift behaviors.

Table 3 and 4 present the average accuracy over the entire stream for the datasets 4CRE-V2 and MG-2C-2D for values of θ from 200 to 1000 and $|\mathcal{T}|$ from 50 to 1200. For each dataset, the results that outperformed *Sliding* setting are in bold. All other values present an intermediate result between *Static* and *Sliding*.

Table 3: Average accuracy for 4CRE-V2 dataset varying the parameters θ – max pool size and $|\mathcal{T}|$ – amount of initial labeled data.

θ								
1000	**91.38**	**91.40**	**91.39**	**91.30**	**91.31**	**91.40**	**91.40**	**91.35**
800	**91.76**	**91.71**	**91.72**	**91.68**	**91.71**	**91.71**	**91.68**	**91.65**
600	**91.89**	**91.88**	**91.90**	**91.91**	**91.87**	**91.87**	**91.88**	**91.85**
500	**91.95**	**91.92**	**91.94**	**91.99**	**91.90**	**91.93**	**91.89**	**91.90**
400	**91.97**	**91.95**	**91.95**	**91.96**	**91.93**	**91.94**	**91.92**	**91.93**
300	**91.91**	**91.88**	**91.94**	**91.87**	**91.88**	**91.92**	**91.85**	**91.86**
200	**91.65**	**91.70**	**91.73**	**91.72**	**91.72**	**91.71**	**91.70**	**91.69**
$\|\mathcal{T}\|$	**50**	**100**	**200**	**400**	**600**	**800**	**1000**	**1200**

Table 4: Average accuracy for MG-2C-2D dataset varying the parameters θ and $|\mathcal{T}|$.

θ								
1000	**92.86**	**92.86**	**92.87**	**92.93**	**92.90**	**92.99**	**92.87**	**92.87**
800	**92.89**	**92.94**	**92.87**	**92.91**	**92.91**	**92.88**	**92.84**	**92.88**
600	**92.91**	**92.85**	**92.91**	**92.85**	**92.86**	**92.86**	**92.85**	**92.84**
500	**92.85**	**92.84**	**92.85**	**92.87**	**92.82**	**92.84**	**92.88**	**92.80**
400	**92.81**	**92.87**	**92.78**	**92.85**	**92.75**	**92.84**	**92.75**	**92.82**
300	**92.72**	**92.74**	**92.79**	**92.73**	**92.71**	**92.73**	**92.71**	**92.65**
200	**92.29**	79.66	80.19	80.17	80.15	80.13	80.11	80.09
$\|\mathcal{T}\|$	**50**	**100**	**200**	**400**	**600**	**800**	**1000**	**1200**

In Tables 3 and 4 we can note in most cases that SCARGC outperforms *Sliding* independently of the parameters values. In general, a choice of a small value for the pool size is safer. However, we note that this size must be sufficiently large to enable enough data to represent the clusters. The amount of initial labeled data, $|\mathcal{T}|$, do not have a significant influence on the results.

Regarding the parameter k, a simple heuristic is to use $k = c$, where c is the number of classes. However, it is not always possible to track certain classes using a single cluster. An example is the MG-2C-2D (Fig. 3), where it was necessary to use 2 clusters for each class.

In the benchmark datasets used in this study, one or two clusters are usually sufficient to track the changes. However, the use of a larger amount of clusters per class can approximate less conventional shapes and track changes in these classes. We evaluated the behavior of SCARGC (for brevity, only with SVM as base classifier) with different values for k in 1CSurr dataset. Fig. 5 gives an intuition of the classes evolution for this data. In this dataset, the classes are generated from Gaussian distributions. The ○ class has a smaller standard deviation than the △ class. The mean of the ○ class moves according to the dotted lines. Some examples from the △ class are removed to reduce the overlap between classes. One of the purposes of this dataset is to simulate a problem with several classes that was transformed into a binary problem. We can consider that the △ class consists of the union of several classes.

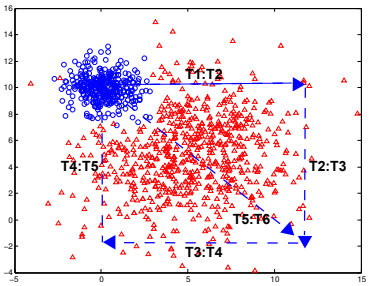


Figure 5: Behavior of 1CSurr dataset. Path performed by the class ○ over time in 5 steps.

The results achieved by SCARGC for 1CSurr considering $k = \{2, 4, 6, 8, 10\}$ can be seen in Fig. 6.

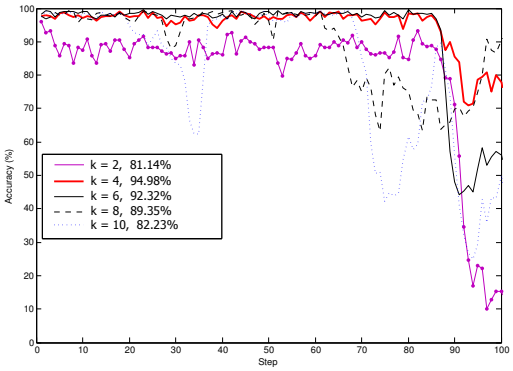


Figure 6: Results varying k for 1CSurr dataset.

The best result is achieved with $k = 4$ with 94.98% of mean accuracy. The algorithm has a certain instability with values different of 4, mainly at the end of dataset, when the class ○ is placed in the center of feature space and the class △ is distributed around it. However, even with this instability the results outperforms the *Static* setting (65.55%).

4.6 Efficiency Analysis In stream processing, the time efficiency is a requirement for many applications.

SCARGC is simple and has low computational cost compared to rival methods. Table 5 shows the wall clock time in seconds averaged over 10 runs for each synthetic dataset used in our experiments. We performed the tests in a 2.4 GHz desktop computer with 8 GB RAM. Our algorithm was implemented in MATLAB.

Table 5: Computation time of SCARGC (in seconds).

	Base classifier			Base classifier	
Dataset	**1NN**	**SVM**	**Dataset**	**1NN**	**SVM**
1CDT	9.51	1.83	1CSurr	32.51	8.78
2CDT	9.63	2.01	4CE1CF	106.95	38.79
1CHT	9.48	1.72	UG-2C-2D	59.76	10.11
2CHT	9.58	1.71	UG-2C-3D	118.41	20.78
4CR	86.77	27.52	UG-2C-5D	120.21	31.18
4CRE-V1	75.69	11.61	FG-2C-2D	119.70	20.25
4CRE-V2	111.42	22.22	MG-2C-2D	117.67	21.37
5CVT	24.48	5.49	GEARS-2C-2D	116.20	15.33

The 1NN classifier has a lazy approach and therefore has no computational cost for updating the classification model, while the SVM classifier requires some time to induce a new model at each update. In contrast, during the classification phase, SVM is usually faster than the 1NN. The final running times of SCARGC using SVM as the base classifier are lower than using 1NN.

A direct comparison of our results with rivals algorithms is not possible for all datasets. However, in [6] is shown the running times for COMPOSE and APT for the datasets UG-2C-2D, UG-2C-3D, and MG-2C-2D. Their experimental setup employs a computational setting very similar to ours. For these datasets, the running times for COMPOSE were respectively 249.6 seconds, 1599.6 seconds and 499.8 seconds. APT spent 3,600 minutes, 22,776 minutes and 20,303 minutes. Thus, the proposed algorithm requires significantly less computational resources than rival methods. For these data, SCARGC spent only 10.11 seconds, 20.78 seconds and 21.37 seconds, respectively, using SVM as base classifier.

4.7 Real World Data We also evaluated the performance of SCARGC in two real world applications related to weather prediction and keystroke dynamics.

The first real data evaluated was compiled by the U.S. National Oceanic and Atmospheric Administration. The NOAA dataset consists of weather measurements collected over 50 years at Bellevue, Nebraska. This dataset has also been used to evaluate COMPOSE and APT in [6]. The NOAA dataset contains eight features: temperature, dew point, sea-level pressure, visibility, average wind speed, max sustained wind speed, minimum temperature, and maximum temperature. The classification task in this dataset is to determine whether it will rain or not. The dataset contains 18,159 daily readings of which 5,698 are *rain* and the remaining 12,461 are *no rain*.

Although in this application is possible to obtain the actual labels of classified examples after a 24 hour lag, we use only labeled examples from first month to predict the weather of the next 50 years.

The results achieved by the *Static, Sliding*, COMPOSE, APT and SCARGC classifiers are shown in Fig. 7. The *Static* setting uses only the first 30 examples. The *Sliding* classifier has knowledge of the actual labels and we keep a window of the last 30 days. SCARGC uses only 10 initial labeled examples and verifies whether it needs an update every month (i.e., the max size of pool data is 30).

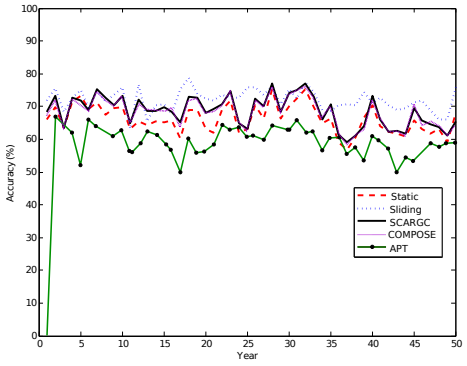


Figure 7: Behavior of SCARGC in the NOAA dataset with $k = 2$. COMPOSE with $CP = 0.65$ and $\alpha = 14$.

SCARGC outperformed the *Static* classifier and it was outperformed by the *Sliding* classifier. The mean accuracy for the entire evaluation was 66.19% for *Static*, 68.61% for SCARGC and 72.01% for *Sliding*. Against rival methods, it is possible to observe that the SCARGC outperformed the APT and showed a very similar behavior to the COMPOSE.

Our second real world problem is based on the use of keystroke dynamics to recognize users by their typing rhythm instead the simple login and password verification. The analysis of this pattern can be used as a second security layer for user authentication without any additional hardware costs. However, the system needs regularly updating the user profile because they evolve incrementally over time as suggested by [2].

We built a stream dataset of keystroke dynamics based on CMU data [8]. In CMU data, 51 users type the password ".tie5Roanl" plus the *Enter* key 400 times captured in 8 sessions performed in different days. To perform the user classification task, we used 10 features extracted from the *flight time* for each pressed key. The *flight time* is the time difference between the instants when a key is released and the next key is pressed. In our stream dataset, we randomly chose 4 users and merged them respecting the chronological order in a

total of 1,600 examples. Fig. 8 presents an example that illustrates the dataset.

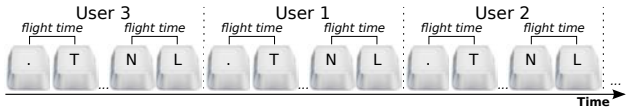


Figure 8: Illustration of the keystroke dynamics dataset.

Considering the task of classifying each one of four possible users over time according to their typing profile, we present in Fig. 9 the results achieved by SCARGC against the lower and upper bounds settings *Static* and *Sliding*. As each user types 50 times per session, we initially train the classifiers with part of data from the first session (first 150 instances) and evaluate the performance of classifiers in the remaining 7 sessions.

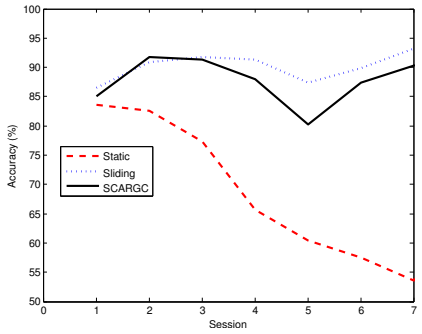


Figure 9: Results of SCARGC ($|\mathcal{T}| = 150$, $\theta = 150$, $k = 12$) against *Static* and *Sliding* (both classifiers with $|\mathcal{T}| = 150$) on the keystroke dynamics dataset.

We can note in Figure 9 that *Static* classifier clearly degrades their performance over time due to changes in the users profiles. SCARGC is able to adapt to changes without the knowledge of actual labels over time. In summary, the *Static* classifier achieves 68.69% of accuracy, *Sliding* achieves 90.14% and SCARGC performs very well with 87.72% of accuracy.

5 Conclusions

This paper presents an efficient and accurate classification method for data streams with incremental drifts in *extreme verification latency* scenario. We performed a wide experimental evaluation over sixteen synthetic and two real world datasets. In all datasets, SCARGC produces accurate classifiers using a small amount of initial labeled data. We also showed that SCARGC is significantly faster than the state-of-the-art methods.

Our future efforts will target the removal of the parameter k by estimating the number of clusters automatically, the use of additional statistics to compare the clusters over time, and deal with emerging classes and disappearance or fusion of classes during the stream.

Acknowledgment

This work was funded by São Paulo Research Foundation (FAPESP) under grant numbers 2011/17698-5, 2012/50714-7, 2013/26151-5, and by European Commission through the project MAESTRA under grant number ICT-2013-612944.

References

[1] Paper website. *https://sites.google.com/site/nonstationaryarchive*.

[2] L. C. Araújo, L. H. Sucupira Jr, M. G. Lizarraga, L. L. Ling, and J. B. T. Yabu-Uti. User authentication through typing biometrics features. *IEEE TSP*, 53(2):851–855, 2005.

[3] A. Bifet and R. Gavalda. Learning from time-changing data with adaptive windowing. In *SDM*, volume 7, pages 443–448, 2007.

[4] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

[5] J. C. Dunn. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *J. Cybernet*, 3:32–57, 1973.

[6] K. B. Dyer, R. Capo, and R. Polikar. Compose: A semisupervised learning framework for initially labeled nonstationary streaming data. *IEEE TNNLS*, 25(1):12–26, 2014.

[7] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *SBIA*, pages 286–295, 2004.

[8] K. Killourhy and R. Maxion. Why did my detector do that?! In *Recent Advances in Intrusion Detection*, pages 256–276, 2010.

[9] G. Krempl. The algorithm APT to classify in concurrence of latency and drift. In *Advances in Intelligent Data Analysis X*, pages 222–233. 2011.

[10] G. Krempl, I. Žliobaite, D. Brzeziński, E. Hüllermeier, M. Last, V. Lemaire, T. Noack, A. Shaker, S. Sievi, M. Spiliopoulou, and J. Stefanowsky. Open challenges for data stream mining research. *ACM SIGKDD Explorations Newsletter*, 16(1):1–10, 2014.

[11] G. R. Marrs, R. J. Hickey, and M. M. Black. The impact of latency on online classification learning with concept drift. In *Knowledge Science, Engineering and Management*, pages 459–469. 2010.

[12] D. Pelleg, A. W. Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML*, pages 727–734, 2000.

[13] V. M. A. Souza, D. F. Silva, and G. E. A. P. A. Batista. Classification of data streams applied to insect recognition: Initial results. In *BRACIS*, pages 76–81, 2013.