Online Clustering of Multivariate Time-series

Masud Moshtaghi, Christopher Leckie and James C. Bezdek*

Abstract

The intrinsic nature of streaming data requires algorithms that are capable of fast data analysis to extract knowledge. Most current unsupervised data analysis techniques rely on the implementation of known batch techniques over a sliding window, which can hinder their utility for the analysis of evolving structure in applications involving large streams of data. This research presents a novel data clustering algorithm, which exploits the correlation between data points in time to cluster the data, while maintaining a set of decision boundaries to identify noisy or anomalous data. We illustrate the proposed algorithm for online clustering with numerical results on both real-life and simulated datasets, which demonstrate the efficiency and accuracy of our approach compared to existing methods.

1 Introduction

Streaming data analysis is a cornerstone of the era of big data. Many new and emerging applications such as mining content from social media, the Internet of Things (IoT), self maintaining sensors, and monitoring systems depend on processing large volumes of streaming data. The main requirements for data analytic methods for these applications are efficiency and the ability to perform adaptive, online unsupervised learning. Clustering algorithms address this problem by finding a finite set of categories that explain the structure of the data stream [19].

Clustering can be a computationally expensive problem when algorithms require multiple passes over the data to find the clusters. Consequently, efficient on line clustering algorithms are difficult to develop. Algorithms for online clustering can be divided into two categories. The first category is general clustering algorithms for any sequence of data. These algorithms do not assume any ordering in the data stream and require the number of clusters to be specified in advance, for example sequential k-means and sequential agglomerative clustering [1]. A second category of online clustering algorithms assume a natural ordering in the data (time-series) and operate based on the assumption that close observations in time will be closely related. These algorithms automatically find the number of clusters using change-point detection techniques. A recent algorithm in this category (called *eClustering+*) provides an incremental approximation of subtractive clustering [4, 5] and features an efficient online learning technique. The main shortcomings of this approach are: (1) it requires a statistically standardized data stream, (2) it has many user-defined parameters, (3) it can only find axis-parallel hyperellipsoidal clusters, and (4) it usually results in a higher than expected number of clusters. These shortcomings result in clusters that are not representative of the regularities in the data, and thus is not suitable for data summarization.

This paper develops a new algorithm for online clustering of time-series that offers three contributions: (1) a clustering algorithm called *Online Elliptical Clustering* (OEC) based on the idea of *Weighted Incremental Data Capture Anomaly Detection* (*wIDCAD*), which was developed in [15]; (2) a fast state-tracking model capable of detecting new clusters using the concept of c-separation; and (3) an empirical evaluation of the performance of the proposed approach on a real-life and two synthetic datasets. Our results demonstrate that our clustering algorithm achieves a higher accuracy than the existing online clustering algorithms reported in [1, 4, 5].

The next section summarizes related work. In Section 3, we present the mathematical details of our approach. In Section 4, we describe the steps of our online clustering algorithm. Section 5 contains numerical evaluation of our method and comparisons to previous approaches. A summary and conclusions are given in Section 6.

2 Background and Related Work

There are two main approaches to clustering in data streams [11]: (1) buffer a window of the data stream, apply efficient batch clustering algorithms to find clusters in the window, and merge clusters of adjacent windows to obtain a final clustering [2, 3, 18]; and (2) incremental learning techniques to find clusters in the evolving data stream [1, 5]. Techniques in the first category leverage the vast amount of existing research on batch clustering but their performance in large data streams suffers from the complexity of multi-pass clustering approaches that

*Department of Computing and information Systems, The University of Melbourne, Australia

are usually used in each window. In addition, selecting an appropriate window size is difficult [17]. The second category of data streaming algorithms, also known as *online clustering* algorithms, process the inputs one at a time and do not need a buffer, thereby providing the efficiency required to analyze large data streams.

Online clustering algorithms can themselves be subdivided into two categories as depicted in Fig. 1: general clustering algorithms [1, 6, 14]; and clustering algorithms for time-series [4, 5]. The general clustering algorithms can be applied to time-series, but they usually require the number of clusters to be specified in advance. In contrast, by assuming the principle of locality in the time-series, algorithms for time-series automatically identify the number of clusters using change-point detection techniques. Therefore, these algorithms are more suitable for time-series but lose their generality by becoming tied to the order of data.

Online clustering algorithms for time-series have two components. The first component is an online change detection algorithm, which identifies changes in the data stream as potential times for creating a new cluster in the data. The second component is an online modeling technique that builds a model of the clusters. In streaming environments, clustering is often used to summarize or compress the data stream into a representative model of the data.

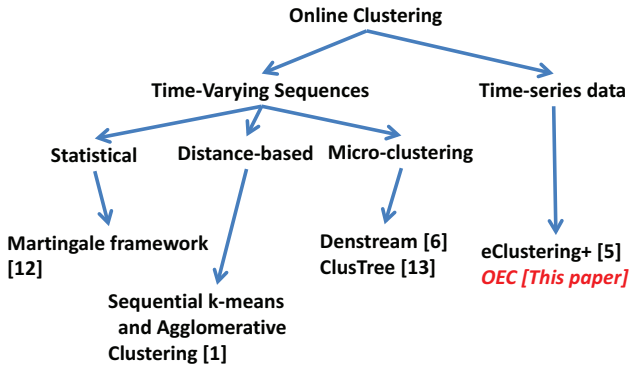


Figure 1: A taxonomy of online clustering algorithms.

Statistical methods for online clustering of time-series involve detecting a concept change in the data by using a statistical test to detect the deviation of the data from its mean [12]. Mozafari *et al.* proposed an online technique based on a Martingale framework [12] to detect changes in the data stream [17]. The detection of a change triggers the creation a new cluster. The detection is based on the violation of the exchangeability condition using a randomized power Martingale. However, the assumption of exchangeability for the auto-correlated sequence of data in time-series is not practical.

The closest model to our approach is the model proposed by Angelov and Zhou [4, 5] called *eClustering+*. eClustering+ is an incremental version of subtractive clustering method proposed by Chiu [7]. The recommended treatment of each input to eClustering+ is to standardize it, feature by feature, which can be problematic in data streams as the range of values might not be known beforehand. This method estimates densities along different axes and ignores correlations between features in the input space, resulting in a large number of clusters.

In this paper, we propose an online clustering algorithm where the change detection part of the algorithm is implemented using the concept of c-separation [9]. An incremental learning technique called *wIDCAD* [15] is used to model the clusters. This learning algorithm is initially proposed to maintain summary information in the form of the mean and inverse of the covariance matrix in the premises of a Takagi-Sugeno fuzzy rule-based system [20]. wIDCAD does not need the features to be normalized, takes correlation between features into account and can operate in noisy data streams, which makes it ideal for online clustering of time-series.

3 Problem Statement and Definitions

The two main questions that need to be answered by an online clustering algorithm for time-series are: (**Q1-New cluster rule:**) When is a new cluster needed to describe the data? And (**Q2-Update rule:**) How to update cluster representations/summaries upon the arrival of a new point? Fig. 2 illustrates the first question using data from a two-state dynamic system. When the system changes its state for the first time, the algorithm starts observing *unusual* data points. At this stage, the algorithm needs to decide whether the data is noise, belongs to the initial cluster or a new cluster is needed. Fig. 3 shows a case where a new point arrives after the clustering algorithm has identified two clusters in the data. For this data point, the clustering algorithm needs to update the existing clusters based on how close the point is to each cluster. Our OEC algorithm incorporates the abilities necessary to answer these two questions.

Let $X_k = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k\}$ be the first k samples at times $\{1, 2, \ldots, k\}$, where each sample is a $p \times 1$ vector in $\Re^p$. The hyperellipsoid of effective radius t centered at the sample mean of the first k data, viz. m_k, with covariance matrix S_k is defined as

$$e_k(m_k, S_k^{-1}; t) = \left\{x \in \Re^p | (\boldsymbol{x} - m_k)^T S_k^{-1} (\boldsymbol{x} - m_k) \leq t^2\right\}. \tag{3.1}$$

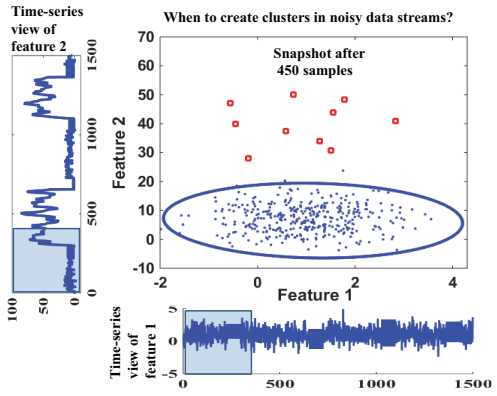


Figure 2: A snapshots of the data after observing a few data points from the second state for the first time.

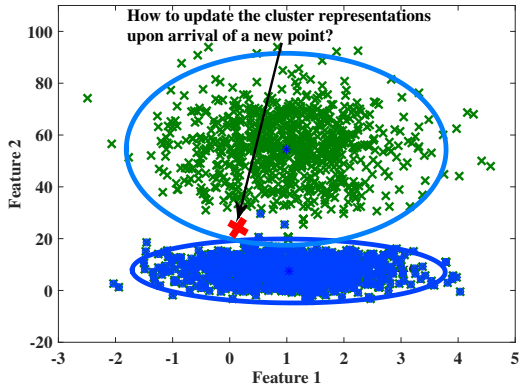


Figure 3: An arrival of a point between the two clusters when both clusters have been identified.

Remark: $(\boldsymbol{x} - m_k)^T S_k^{-1}(\boldsymbol{x} - m_k)$ is the Mahalonobis distance from x to m_k and S_k^{-1} is the matrix of the hyperellipsoid e_k. We use m_k and S_k^{-1} to represent a hyperellipsoidal cluster at time k. The boundary of the hyperellipsoidal cluster e_k is defined as

$$\delta_{e_k}(m_k, S_k^{-1}; t) = \left\{ x \in \Re^p | (\boldsymbol{x} - m_k)^T S_k^{-1}(\boldsymbol{x} - m_k) = t^2 \right\}. \tag{3.2}$$

We define an *anomaly* with respect to e_k as any data vector $\boldsymbol{x} \in \Re^p$ that is outside e_k:

x is *anomalous* for $e_k \Leftrightarrow$

$$(\boldsymbol{x} - m_k)^T S_k^{-1}(\boldsymbol{x} - m_k) > t^2. \tag{3.3}$$

We choose $t^2 = (\chi^2)_p^{-1}(\gamma)$ (i.e., the inverse of the chi-squared statistic with p-degrees of freedom). This results in an ellipsoid that covers at least $100\gamma\%$ of the data under the assumption that the data has a Gaussian distribution [21]. The Gaussian assumption is rarely true in time-series. However this threshold is a close approximation for any unimodal distribution.

4 Online Elliptical Clustering (OEC)

In this section, we explain how the wIDCAD update formulas are used to update cluster representations at time $k+1$ (the update rule), then we introduce the state-tracker that is used to identify when to create a new cluster, and finally we present the algorithm.

4.1 Weighted IDCAD In [15], the authors introduced incremental update formulas for the mean and the inverse of the covariance matrix where some weight ω_k is attached to each data point. We use these update formulas to update our ellipsoidal cluster representations for each new data point $\boldsymbol{x}_{k+1}$. The weights in our algorithm are the normalized degree of membership of $\boldsymbol{x}_{k+1}$ to each cluster, which are denoted by $\omega_{k+1,i}$, $1 \le i \le |C|$, where C is the set of crisp clusters with boundaries specified by (3.2) at time k ($\sum \omega_{k+1,i} = 1$). Since OEC assigns weights to each sample, it can be used as a soft (fuzzy) clustering algorithm. However, we consider a crisp version of OEC in this paper. We use the following formula to incrementally update cluster representations.

$$m_{k+1,i} = m_k + \frac{\omega_{k+1,i}}{\sum_{j=1}^{k+1} \omega_{ji}}(\boldsymbol{x}_{k+1} - m_{t,i}); 1 \le i \le |C|. \tag{4.4}$$

$$S_{k+1,i}^{-1} = \chi_{ki} \times \left[S_{ki}^{-1} - \frac{S_{ki}^{-1}(\boldsymbol{x}_{k+1} - m_{ki})(\boldsymbol{x}_{k+1} - m_{ki})^T S_{ki}^{-1}}{\delta_{ki} + (\boldsymbol{x}_{k+1} - m_t)^T S_{k,i}^{-1}(\boldsymbol{x}_{k+1} - m_{k,i})} \right], \tag{4.5}$$

where

$$\alpha_{k,i} = \sum_{j=1}^{k} \omega_{j,i}$$

$$\beta_{k,i} = \sum_{j=1}^{k} \omega_{j,i}^2$$

$$\chi_{k,i} = \frac{\beta_{k,i}(\beta_{k+1,i}^2 - \alpha_{k+1,i})}{\beta_{k+1,i}(\beta_{k,i}^2 - \alpha_{k,i})}$$

$$\delta_{k,i} = \frac{\beta_{k+1,i}(\beta_{k,i}^2 - \alpha_{k,i})}{\beta_{k,i}\omega_{k+1,i}(\beta_{k+1,i} + \omega_{k+1,i} - 2)}$$

It is important to protect the incremental algorithm against noise and large-valued outliers. The definition in (3.3) allows the algorithm to identify anomalies. In [15], the authors suggested using two values for γ, $\gamma_1 = 0.99$ for the normal boundary of each cluster; and $\gamma_2 = 0.999$ to provide a *guard-zone* around the normal boundary. The guard-zone protects the incremental

algorithm against large-valued outliers. An update rule updates the sample mean and the inverse of the covariance matrix when a new point arrives unless the point is outside the guard-zone. To ensure that the boundaries in (3.3) have reasonable statistical power, we consider a *stabilization period* n_s for our incremental algorithm. The guard-zone for a cluster is only activated after the stabilization period. The boundary defined by $\gamma_1 = 0.99$ can be used to mark anomalies with respect to the normal boundaries of the clusters. However, in this paper, we will not evaluate the capabilities of the algorithm in identifying anomalies.

Upon arrival of $\boldsymbol{x}_{k+1}$, we first calculate the set of $|C|$ Mahalonobis distances, say $\{M_{k+1,i} : 1 \leq i \leq |C|\}$ from $\boldsymbol{x}_{k+1}$ to the $|C|$ current cluster centers $\{m_{k,i}\}$. Then, we test the data point against definition (3.3) and the ellipsoidal thresholds defined by γ_2. Let $C' \subseteq C$ be the set of clusters that have the new point in their guard-zone. The weights associated with the new data point for each cluster C_i in C' are calculated using

$$\omega_{k+1,i} = \frac{e^{\frac{-1}{2}M_{k+1,i}}}{\sum_{j=1}^{|C'|} e^{\frac{-1}{2}M_{k+1,j}}}. \tag{4.6}$$

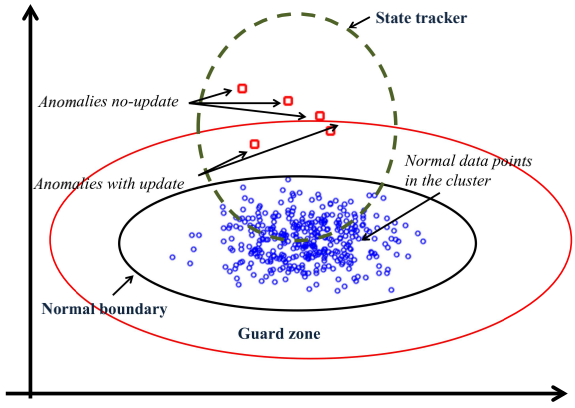


Figure 4: The initial cluster with its guard-zone and the state-tracker at the start of a change of the state of the system (same dataset as Fig. 2).

4.2 State Tracker Identifying new clusters in time-series is based on change-point detection. Most change-point detection techniques rely on a user-specified threshold to differentiate between a change in the state of the system or normal variability within a state. *Our aim is to identify new clusters without the need for a user-specified threshold.* This is accomplished by introducing a state-tracker to track the state of the system using a single cluster model with a forgetting factor. The forgetting factor allows the model to respond quickly to changes in the data stream - if the state-tracker is significantly different from the existing clusters we will create a new cluster. Fig. 4 shows how the state-tracker follows the data points after the change point, while the initial cluster, which is protected by the guard-zone and does not use forgetting, stays in the first state of the system.

We use the incremental formulas defined in [16] to update the parameters of the state-tracker, $m_{k+1,\lambda}$ and $S^{-1}_{k+1,\lambda}$ at time $k+1$ with forgetting factor λ. These parameters can be estimated using equations (4.7) and (4.8). The suggested range for λ in the estimation theory literature is [0.9, 1) [16].

$$m_{k+1,\lambda} = \lambda m_{k\lambda} + (1-\lambda)\boldsymbol{x}_{k+1} \tag{4.7}$$

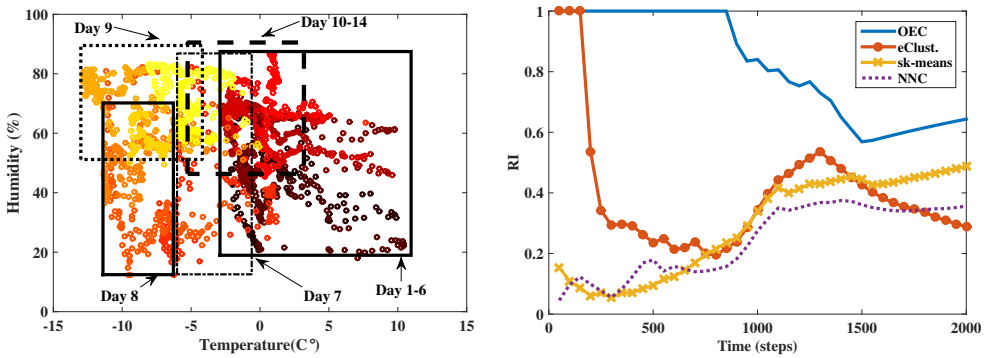
$$S^{-1}_{k+1,\lambda} = \frac{kS^{-1}_{k\lambda}}{\lambda(k-1)} \times \left[I - \frac{(\boldsymbol{x}_{k+1} - m_{k\lambda})(\boldsymbol{x}_{k+1} - m_{k\lambda})^T S^{-1}_{k\lambda}}{\frac{(k-1)}{\lambda} + (\boldsymbol{x}_{k+1} - m_{k\lambda})^T S^{-1}_{k\lambda}(\boldsymbol{x}_{k+1} - m_{k\lambda})} \right] \tag{4.8}$$

The formula (4.8) is not an exact estimation of an equivalent batch update. As more samples are processed, the growth of k dominates (4.8). To control this, we cap the value of k to its *effective* value based on the *memory horizon* of the tracker [16]. After the arrival of $N_{eff} = 3\tau$ samples, we keep k constant at N_{eff}, where $\tau = \frac{1}{1-\lambda}$ is known as the *memory horizon* of an iterative algorithm with an exponential forgetting factor λ.

Now, we are ready to define our *new-cluster rule*, which is based on the *c-separation* measure defined by Dasgupta [8, 9] in the context of learning Gaussian mixtures. Two p-variate Gaussian densities $N_p(\mu_i, \Sigma_i)$ and $N_p(\mu_j, \Sigma_j)$ are said to be c-separated if

$$\|\mu_i - \mu_j\| \geq c\sqrt{p \times max\left(d_{max}\left(\Sigma_i\right), d_{max}\left(\Sigma_j\right)\right)}, \tag{4.9}$$

where $d_{max}(\Sigma)$ is the largest eigenvalue of Σ, and the value of $c \in [0, \infty)$ specifies the degree of separation between the two distributions. According to Dasgupta, a 2-separated mixture corresponds to almost completely separated Gaussian distributions. Using this idea, ***when the tracker becomes 2-separated from the existing clusters, we create a new cluster***. This normally co-occurs with a burst of consequent anomalies. Therefore, to provide information about the start of the change in the data stream, we keep a buffer of consequent anomalies using the $\gamma_1 = 0.99$ boundary,



(a) Scatter plot of the real-life dataset - Sensor 18 (b) Rand-Index values at different steps

Figure 5: Left: Scatter plot of the real dataset used in the evaluation. The plot shows the progression of time, starting with black becoming lighter with time. Right: RI values calculated at different snapshots of the algorithms.

which is cleared after a normal observation. After creating a new cluster, the start of the buffer is flagged by the algorithm as the change-point, and the data inside the buffer is used to create the new cluster. However, checking for 2-separation between the tracker and the normal clusters can be computationally expensive if performed for every data point. Therefore, we start checking for 2-separation between the tracker and the normal clusters if there are at least $p+1$ consequent anomalies in the buffer. Note that we need at least $p+1$ unique points to calculate an invertible covariance matrix in p-dimensions.

4.3 OEC Algorithm The first step of the algorithm is to calculate the Mahalanobis distance of the new point to the center of each cluster to determine whether the point is considered anomalous for any clusters and if the corresponding cluster should be updated. In the second step, the clusters and the state-tracker are updated using the appropriate formulas. Finally, if there is a need for a new cluster, a cluster is formed using the consequent anomaly buffer. The pseudo-code of the algorithm is given in the supplementary materials.

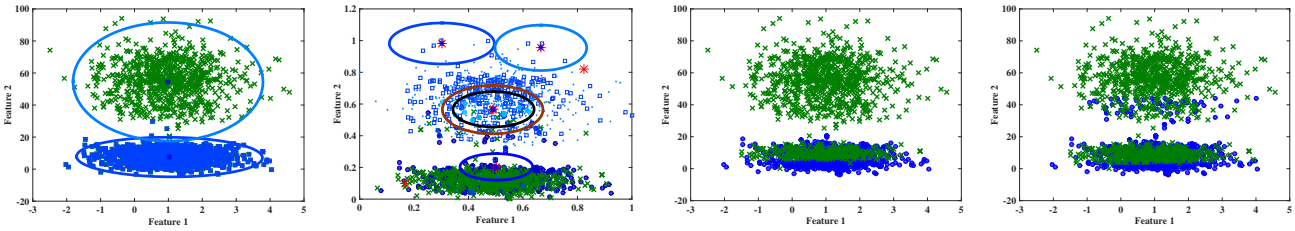
5 Evaluation

In this section, we compare our algorithm with the eClustering+ algorithm detailed in [4, 5], sequential k-means (*sk-means*) and agglomerative nearest-neighbor clustering (*NNC*). We start by describing the settings used for each algorithm, then we discuss the performance of the algorithms on two synthetic and a real-life weather monitoring dataset. All the algorithms are compared using both visual inspection and two external cluster validity indeces called the *Rand Index* (RI) and *Fowlkes-Mallows* [13]. Since both indexes showed similar trends we only show the RI graphs in the paper and refer the reader to supplementary materials for Fowlkes-Mallows graphs. External validity indexes require ground truth labels. In our real-dataset, we use information about the experiment to label data based on significant weather phenomena during the monitoring period. Additionally, OEC and eClustering+ are compared in terms of the identified number of clusters.

Since it is difficult to convey a real sense of how the algorithm works with static images, a video from the snapshots of OEC in these datasets is made available online at (`https://youtu.be/dZi9eVQKTNQ`).

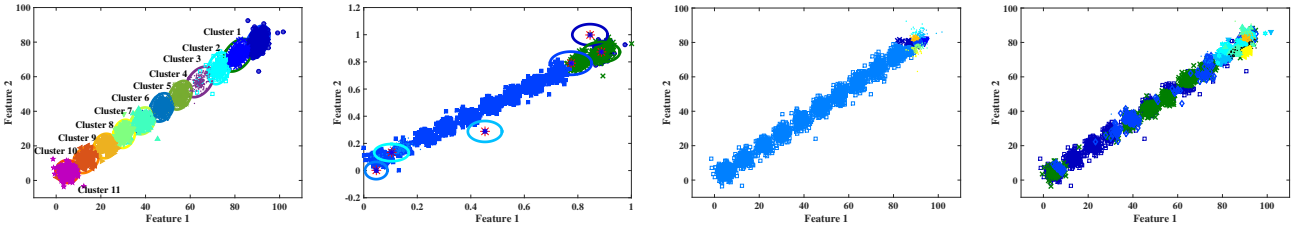
5.1 Parameter Settings eClustering+ requires a statistically standardized dataset. Since incremental standardization may lead to unsatisfactory results [15], we give eClustering+ a head start by feeding it with *batch* normalized data. In reality, this can only be done when the range of the values for each feature is known prior to the start of the algorithm. The parameters of the eClustering+ algorithm are set according to the recommendations in [4]. Since we are seeking to find a set of clusters that represent the whole stream, we do not use the cluster disabling feature of this algorithm. The parameter settings for OEC are the use of 2-separation, a stabilization period of $n_s = 20$, and effective radii $\gamma_1 = t_1^2 = (\chi^2)_p^{-1}(0.99)$ and $\gamma_2 = t_2^2 = (\chi^2)_p^{-1}(0.999)$ for the normal cluster boundary and the guard zones. The true number of clusters in synthetic datasets and the believed number of clusters in the real dataset are given to sk-means and NNC algorithms as they need *a priori* knowledge of the number of clusters.

5.2 Dataset S1: Synthetic Locally Linear Processes The first dataset comprises a switching time-



(a) OEC (b) eClustering+ (c) sk-means (d) NNC

Figure 6: The final set of clusters in $S1$.



(a) OEC (b) eClustering+ (c) sk-means (d) NNC

Figure 7: The final set of clusters in $S2$.

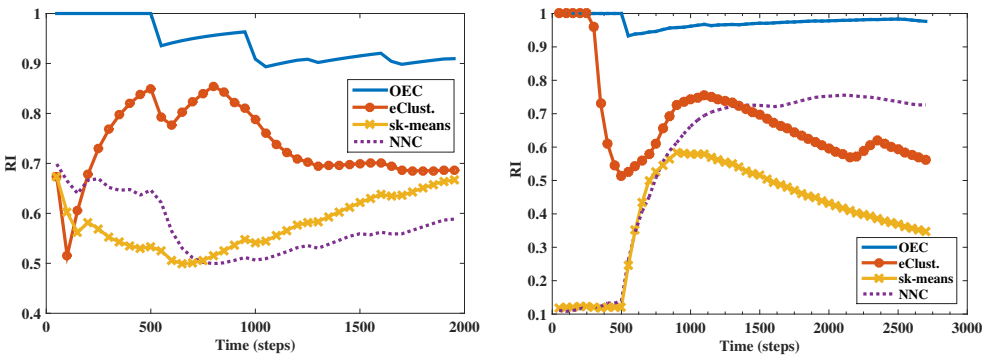
series with two different modes. The time-series changes four times between its two modes. This dataset consists of two well-separated clusters. The supplementary material contains more details about this dataset.

The final clusters found by the algorithms are shown in Fig. 6. Note that the axes of the graphs for eClustering+ are all between [0, 1] as standardized values are fed to the algorithm. OEC finds the two clusters in this dataset and summarizes them accurately by the two decision boundaries. The eClustering+ algorithm, despite the axis-parallel nature of this dataset, finds 5 clusters, which is significantly higher than the expected 2 clusters. Initially, sk-means and NNC mis-label the data as they try to maintain two clusters in the data while there is only one cluster in the data. As time progresses and the second cluster appears in the data, they distinguish the two clusters successfully. The only difference between the two algorithms is in the way they label the transitional points between the two clusters.

To compare the performance of the algorithms in terms of the RI, we calculate the RI value at different steps in time for the observed samples up to that point. As more samples are labeled by the algorithm the RI value for the observed set might change. The Rand index takes values between 0 and 1. The value 0 indicates no pair-based matches between the generated partition and the ground-truth. The value 1 indicates a perfect match between the two partitions being compared. Fig. 8(a) shows the Rand Index values at different steps in time. OEC clearly outperforms the other methods with respect to this index. eClustering+ after a small period of time performs better that sk-means and NNC. Towards the end of the experiment, the performance of sk-means and NNC increases slightly and approaches the eClustering+ result. Although eClustering+ finds extra clusters in this dataset, RI values show that this algorithm was relatively successful in separating the two modes of this dataset.

5.3 Dataset S2: Shifting Gaussian Distribution

This dataset is a 2-dimensional time-series that initially comes from a Gaussian distributions and this Gaussian distribution changes constantly over time and does not switch back to a previously known Gaussian during the experiment. The constant shift in the distribution is challenging for online clustering algorithms. More details about the dataset are provided in the supplementary materials. The final clusters found by the algorithms are shown in Fig. 7. OEC is the only algorithm which identifies the clusters correctly and its performance in terms of the RI is superior to the other algorithms (Fig. 8(b)). NNC's and eClustering+'s performance are mixed and can be considered as a tie for the second best algorithm. eClustering+ finds 6 clusters and puts 7 of the Gaussian distributions in the middle of the experiment into one cluster. However, it initially performs better in terms of the RI compared to NNC before it loses the lead towards the end the experiment.



(a) Dataset S1 (b) Dataset S2

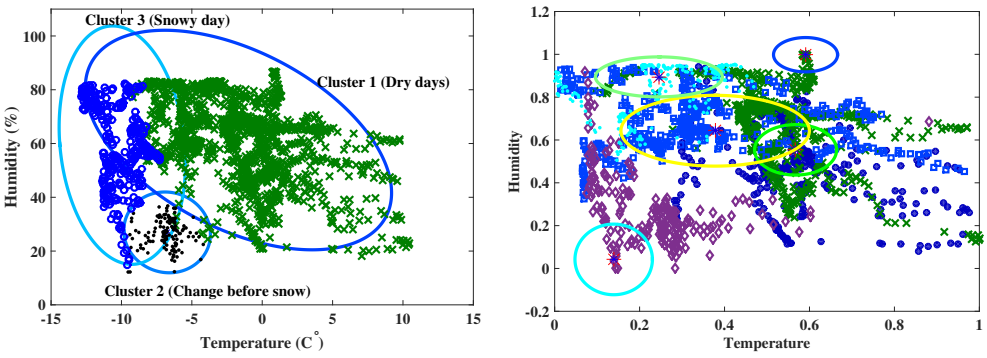
Figure 8: RI values calculated at different snapshots of the the algorithms.

5.4 Dataset 3: Real-life dataset The real-life dataset used in our evaluation is from a collection of weather station nodes in the *Le Genepi* (LG) region in Switzerland [10]. Two weeks of data at node 18 starting from October 10th 2007 are used in the evaluation. We use average surface temperature and humidity readings of node 18 over 10-minute intervals to form two dimensional input vectors $\{\boldsymbol{x}_i\}$. Fig. 5(a) shows the scatter plot of the data. The scatter plot does not offer much visual evidence about the number of clusters in the LG data. However, the imagery information from the site shows that there is a snowy day during the two weeks of data collection and the data confirms that a cold and windy day precedes the snow. Therefore, we consider three clusters in the data: sunny days before and after the snow, cold front moving in, and the snowy day and label data accordingly, i.e., all of the points for days 1-6 and 10-14 are in cluster 1, days 7 and 8 in cluster 2, days 9 in cluster 3. We use k=3 as the number of clusters for sk-means and NNC in this dataset.

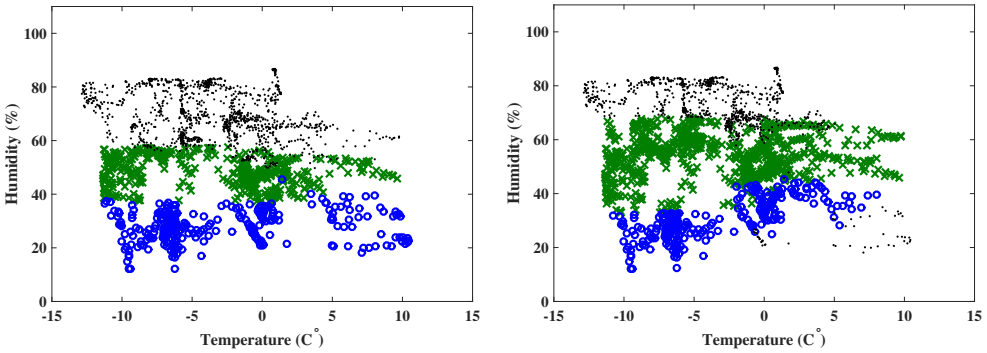
OEC finds the expected number of clusters, while eClustering+ finds a higher number of clusters and mislabels some portion of the data by losing its ability to differentiate between two or more clusters as time goes on. Fig. 9 shows the final clusters found by the four algorithms. The clusters identified by OEC match the ground truth. This is evidenced by the RI values plotted in the Fig.5(b). OEC achieves the best RI values during the experiment, with sk-means perhaps characterized as the second best considering its performance towards the end. However, the cluster structure in OEC matches the evolution of the data over time while sk-means is only trying to achieve maximum separation in the data. In this dataset, the cluster associated with the cold front entering the region has good separation with the other two clusters, therefore sk-means appropriately identifies the cluster. The sunny and snowy clusters are interleaved, hence it is expected that any clustering algorithm will make some misclassification at the boundaries of the clusters. The drop in the performance of the OEC algorithm towards the end of experiment is due to this fact. However, sk-means totally mixes the two clusters. A soft ground-truth labeling of the data would show the differences between the algorithms around the confusion points. However, a reasonably accurate manual soft labeling is not possible with the available information.

5.5 Discussion Online clustering can identify structures in the data stream that traditional batch approaches cannot find. For example, traditional batch clustering algorithms will consider the LG dataset (Fig. 5(a)) as having one cluster. General-purpose online clustering algorithms also suffer from the same weakness as batch clustering because their objective is to achieve maximum separation. However, online clustering algorithms for time-series, directed by change-point detection, identify the evolution of data streams over time.

An important aspect of any unsupervised algorithm is ease of parameter selection. Angelov and Zhou [5] suggested a set of values for the parameters of their incremental algorithm, but these values work only on standardized data. The parameters of our algorithm have statistical interpretations, which simplifies their selection. Moreover, the suggested values used by OEC in this paper are not tied to a specific range of input values and are expected to work in a wide range of datasets. Our experiments (detailed in the supplementary materials) shows that OEC has a very low sensitivity over a wide range of values for the input parameters. The supplementary materials have more information on the sensitivity analysis and parameter selection in OEC.



(a) OEC (b) eClustering+



(c) sk-means (d) NNC

Figure 9: The final set of clusters in the real-life dataset.

The OEC algorithm can also identify anomalies during the process of clustering. For example, if in S2 we used the γ_1 boundary to detect anomalies, we would have detected 50% of the noisy samples and generated about 2% of false alerts. In the future, we plan to investigate this capacity of the algorithm.

OEC has produced better results than the other three clustering algorithms over the three datasets used in this paper. But it is important to highlight the limitations of the OEC algorithm. Similar to eClustering+, OEC is online clustering for time-series, so it is sensitive to the order of the data. Both OEC and eClustering+ only work on real-valued data streams. We believe that if the potential clusters are highly intertwined, for example day-time and night-time in the LG dataset, the OEC algorithm will not be able to differentiate between the two. Finally, if the changes occur slowly the algorithm might just adapt to them rather than creating new clusters, this is something that happened to eClustring+ in S2 but not to OEC. A smaller value of c for c-separation in OEC forces the algorithm to create new clusters for smaller changes. In this paper $c = 2$ is selected so that the algorithm finds well-separated clusters. The supplementary materials contains OEC sensitivity analysis plots for this parameter.

5.6 Complexity Analysis The algorithms discussed in this paper have the same asymptotic time and space complexity, i.e., $O(n)$ and $O(1)$ respectively. The exact time and space complexity of the algorithms for time-series are hard to calculate as they depend on characteristics of the data streams, such as the number of modes and the speed and frequency of the changes. Therefore, we only compare the exact computational cost of a single cluster update. In the exact complexity analysis, all operations and storage space are counted as unit costs. We do not assume economies that might be realized by special programming tricks or properties of the factors involved.

For OEC the computational cost of updating a single cluster is incurred by the operations in (4.4) and (4.5). The number of operations in these two equations is $7p^2 + 7p + 18$. For eClustering+ updating a cluster requires $10p + 18$ operations [15]. Sk-means requires $2p + 2$ to update the centroid of the cluster. Sk-means has the lowest complexity of the updates compared to the other two algorithms. The NNC collectively updates all the centroids. Therefore, we do not attempt to compare it with other algorithms. The main difference between the other three algorithms is the factor p^2 in the OEC single cluster updates. This difference is because

of the matrix operations in OEC versus the vector operations in eClustering+ and sk-means. In most time-series problems, p is small, probably less than 10 in most cases, so this is not an important disadvantage for OEC. Note that the structure of equations (4.4) and (4.5) reduces the complexity of the operations from $O(p^3)$ to $O(p^2)$ by changing two-matrix multiplications to vector-matrix multiplications. All the algorithms have additional costs at each step to calculate the distance of the new data points to each cluster. OEC and eClustering+ have extra complexity at each step to detect new clusters.

6 Conclusions

We have proposed a fully online clustering algorithm based on the idea of incrementally updating a set of ellipsoidal clusters. We used well-studied concepts in statistics and machine learning to select user input parameters compared to algorithm-specific parameters. This simplifies parameter selection for the algorithm. Our comparison of the proposed clustering algorithm to an existing online clustering algorithm for time-series and two online clustering algorithms for general data showed that our algorithm performs significantly better on two synthetic and one real-life datasets.

One way to improve OEC is by incorporating the concepts of ageing and forgetting normal clusters. This will be a focus of our future efforts.

7 Acknowledgment

This research was supported under Australian Research Council's *Discovery Projects* funding scheme (project number DE150100104).

References

[1] M. Ackerman and S. Dasgupta. Incremental clustering: The case for extra clusters. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, pages 307–315. Curran Ass., 2014.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. On clustering massive data streams: A summarization paradigm. In C. C. Aggarwal, editor, *Data Streams: Models and Algorithms*, volume 31, pages 9–38. Springer US, 2007.

[3] N. Ailon, R. Jaiswal, and C. Monteleoni. Streaming k-means approximation. In *Neural Information Processing Systems 2009*, volume 22, pages 10–18, 2009.

[4] P. Angelov. *Evolving Takagi-Sugeno Fuzzy Systems from Streaming Data (eTS+)*, pages 21–50. John Wiley & Sons, Inc., 2010.

[5] P. Angelov and X. Zhou. Evolving fuzzy-rule-based classifiers from data streams. *IEEE Transactions on Fuzzy Systems*, 16(6):1462–1475, 2008.

[6] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SIAM Conf. on Data Mining*, pages 328–339, 2006.

[7] S. Chiu. Fuzzy model identification based on cluster estimation. *Journal of Intelligent and Fuzzy Systems*, 2(17):267–278, 1994.

[8] S. Dasgupta. Learning mixtures of gaussians. In *40th Annual Symposium on Foundations of Computer Science*, pages 634–644, 1999.

[9] S. Dasgupta. Experiments with random projection. In *Proceedings of the Conf. on Uncertainty in AI*, pages 143–151, Stanford, CA, 2000.

[10] SensorScope. `http://lcav.epfl.ch/page-86035-en.html`, 2007.

[11] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transaction on Knowledge and Data Engineering*, 15(3):515–528, 2003.

[12] S.-S. Ho. A martingale framework for concept change detection in time-varying data streams. In *Proceedings of the Int. Conf. on Machine Learning*, pages 321–327, 2005.

[13] L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

[14] P. Kranen, I. Assent, C. Baldauf, and T. Seid. The clustree: Indexing micro-clusters for anytime stream mining. *Knowledge and Information Systems*, 29(2):249–272, 2011.

[15] M. Moshtaghi, J. C. Bezdek, C. Leckie, S. Karunasekera, and M. Palaniswami. Evolving fuzzy rules for anomaly detection in data streams. *IEEE Transactions on Fuzzy Systems*, 2014.

[16] M. Moshtaghi, C. Leckie, S. Karunasekera, J. C. Bezdek, S. Rajasegarar, and M. Palaniswami. Incremental elliptical boundary estimation for anomaly detection in wireless sensor networks. In *IEEE Int. Conf. on Data Mining*, December 2011.

[17] N. Mozafari, S. Hashemi, and A. Hamzeh. A statistical approach for clustering in streaming data. *Artificial Intelligence Research*, 3:38–45, 2014.

[18] M. Salehi, C. Leckie, M. Moshtaghi, and T. Vaithianathan. A relevance weighted ensemble model for anomaly detection in switching data streams. In *Proceedings of PAKDD*, pages 461–473, 2014.

[19] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. de Carvalho, and J. ao Gama. Data stream clustering: A survey. *ACM Computing Surveys*, 46(1):13–31, July 2013.

[20] T. Takagi and M. Sugeno. Fuzzy Identification of Systems and Its Applications to Modeling and Control. *IEEE Transactions on Systems, Man, and Cybernetics*, 15(1):116–132, 1985.

[21] D. M. Tax and R. P. Duin. Data description in subspaces. *Int. Conf. on Pattern Recognition*, 2:672–675, 2000.