Online Cluster Validity Indices for Performance Monitoring of Streaming Data Clustering

M. Moshtaghi[‡] J.C. Bezdek[*] S. M. Erfani[*]
C. Leckie[*] J. Bailey[*]

September 20, 2018

1 abstract

Cluster analysis is used to explore structure in unlabeled batch data sets in a wide range of applications. An important part of cluster analysis is validating the quality of computationally obtained clusters. A large number of different internal indices have been developed for validation in the offline setting. However, this concept cannot be directly extended to the online setting because streaming algorithms do not retain the data, nor maintain a partition of it, both needed by batch cluster validity indices. In this paper, we develop two incremental versions (with and without forgetting factors) of the Xie-Beni and Davies-Bouldin validity indices, and use them to monitor and control two streaming clustering algorithms (sk-means and online ellipsoidal clustering), In this context, our new incremental validity indices are more accurately viewed as performance monitoring functions. We also show that incremental cluster validity indices can send a distress signal to online monitors when evolving structure leads an algorithm astray. Our numerical examples indicate that the incremental Xie-Beni index with forgetting factor is superior to the other three indices tested.

[*]School of Computing and Information Systems The University of Melbourne, AUSTRALIA, [‡] Amazon, Manhattan Beach, California (This work was done before joining Amazon) E-mail: mmasud@amazon.com, jbezdek@unimelb.edu.au, sarah.erfani@unimelb.edu.au, caleckie@unimelb.edu.au, jbailey@unimelb.edu.au

2 Introduction

The intrinsic nature of streaming data requires algorithms that are capable of fast data analysis to extract knowledge. Online clustering algorithms provide a way to extract patterns from continuous data streams. Therefore, online clustering has gained popularity in applications involving massive streams of data, such as router packet analysis and environmental sensing.[1,2] In these applications, the velocity and volume of data is too high for the processing unit to access each data sample more than once. A category of fast online clustering algorithms, also referred to as sequential clustering, process data quickly and efficiently by receiving samples one at a time and updating cluster statistics (such as prototypes) with each sample.[3–5]

An important aspect of traditional clustering algorithms is assessment of the quality of the resulting clusters, i.e., how well does any partition match the input data? *Cluster validity indices* (CVIs) comprise computational models and algorithms whose job is to identify the "best" member among different partitions of batch input data. Most CVIs are max-optimal or min-optimal, meaning that the partition preferred by the index is indicated by the maximum (minimum) value of the index on the partitions being evaluated. To date, CVIs have been used exclusively in a static (or batch) setting, being applied to sets of partitions generated by different parameter settings of the clustering algorithm applied to a collected set of data. In contrast, the question of how well structure is detected by streaming algorithms is quite different. The key assumption in the online context is that data are *processed once*, and *historical data will not be available* for a retrospective analysis. Thus, at the end of online processing, there are no "clusters" retained to examine with a batch CVI, so the question of "cluster validity" for streaming clustering algorithms is a misnomer. Indeed, while the term "streaming or online clustering" is widely used, it is very misleading. Why? Streaming clustering leaves behind only a footprint of its computational history, usually in the form of a set of cluster centers, possibly annotated with time

of creation. The footprint may also include a set of covariance matrices that summarize the structure observed in passing.

This study concerns itself with the use of internal *incremental cluster validity indices* (iCVIs) (streaming data will not have the ground truth partition required by external indices), that are computed online (on the fly) and used to control/interpret clustering for streaming data. It is not our aim to "validate" evolving structure with iCVIs. Instead, we will show how iCVIs can be used to understand and monitor the progress of streaming clustering algorithms. In this context we are essentially using iCVIs to alert us to changes in the input stream.

Most batch indices assess two basic characteristics of a set of clusters: compactness (or density) and separation of the clusters.[6] Compactness is usually calculated based on the observations while separation is often measured by the distance between cluster prototypes. In the online setting, where each observation can be accessed only once, an incremental/recursive calculation of compactness is necessary. In this paper, we propose two incremental methods to estimate within cluster dispersion: (1) an exact incremental update of a batch formula; and (2) an online formula incorporating an exponential forgetting factor. Using these two methods we then derive online versions of two well-known CVIs namely, the *Xie-Beni* (XB)[7] and the *Davies-Bouldin* (DB)[8] indices. These indices can be applied to both hard and soft partitions.

This paper offers four main contributions: (1) we propose a new concept of incremental monitoring of online clustering algorithms which provides new insights into these algorithms; (2) we propose two incremental versions of within cluster dispersion, viz., with and without forgetting, a facility that enables the iCVI to gracefully forget earlier inputs; (3) we propose incremental versions of two well-known batch CVIs allowing exact calculation of the two indices with fast sequential processing of data; and (4) we analyze and discuss the properties of the proposed iCVIs within the context of two online clustering algorithms. Our results demonstrate that iCVIs can provide useful insights

into online clustering algorithms. Moreover, the proposed iCVIs can indicate learning difficulties experienced by the clustering algorithm and can signal the appearance of new clusters in the evolution of the data stream.

The next section summarizes related work. In Section 4, we present definitions and notation needed in this paper. Section 5 contains background information on two important online clustering algorithms. In Section 6, we derive two versions of the iXB and iDB indices and analyze their characteristics. Section 8 contains numerical examples used to evaluate the proposed models. A summary and conclusions are given in Section 9.

3 Background and Related Work

In this section, we briefly describe related work in cluster validation and online clustering algorithms. It is especially important to record our definition of an online clustering algorithm as our goal is to analyze these algorithms using cluster validity indices.

According to Guha et al.,[1] clustering in data streams can be divided into two main strategies: (1) buffering a window of streamed inputs, finding clusters in the window using a batch algorithm such as the classic k-means algorithm; and then merging clusters in adjacent windows to obtain a final clustering. This strategy is espoused, for example, in;[9–11] (2) using incremental learning techniques to find clusters in the evolving data stream. Examples of this strategy include.[3,5,12–15] We refer to this second approach as online or incremental clustering. Algorithms for online clustering can themselves be divided into two sub-categories. The first category is general clustering algorithms for any sequence of data (we refer to this as *type 1* algorithms). 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, or more briefly, sk-means, and sequential agglomerative clustering.[3] A second category of online clustering algorithms assume a natural ordering in the data (time-series) and operate

on the assumption that close observations in time will also be closely spatially. These algorithms use this assumption to dynamically create clusters in evolving data streams (we call these *type 2* algorithms). In this paper, we propose cluster validation methods for online clustering algorithms of both types 1 and 2.

Cluster validity indices (CVIs) can be grouped into two categories: *internal* and *external* indices. Internal indices use only the information available from the algorithmic outputs and the observed unlabeled data. In contrast, external CVIs use additional external information about substructure in the data, usually in the form of a reference partition (a ground truth partition), so the data processed are labeled. External CVIs are used to compare partitions obtained by a clustering algorithm to ground truth labels. Another use of external CVIs is to correlate external and internal assessments of labeled data.[6] In this application the external CVI becomes a tool for selection of the "best" internal CVI when unlabeled data are to be clustered. The use of external CVIs to choose a "good" internal CVI is comprehensively discussed in.[6] Our focus in this paper is on internal CVIs. Milligan and Cooper's 1985 paper is widely regarded as a landmark study for the comparison of internal CVIs.[16]

Internal cluster validity indices fall under the broad umbrella of goodness-of-fit measures such as likelihood ratio tests and Root Mean Squared Error.[17] The majority of goodness-of-fit measures target parametric models while CVIs provide a non-parametric mechanism to evaluate clustering outputs. Another differentiating point between internal CVIs and goodness-of-fit measures is the meaning of the fitness term. Most internal CVI models have components that attempt to capture cohesion and separation, while goodness of fit indices usually assess the fit of a model to the data that generates it.

There are two categories of internal CVIs based on the way that they measure cohesion and separation. The CVIs in the first category use only the partitions generated by the clustering to determine the quality of the partition. Measures of this type include the partition coefficient and partition entropy.[18] Indices such as these often appear in the context of fuzzy cluster validity. However,

most CVIs fall into the second category, that is, they use both the data and the partition information.

We develop incremental CVIs by deriving an incremental formula for the data-dependent part of two well-known indices, i.e., XB and DB. After determining the incremental update formula for cluster cohesion, we derive one step update formulae for these two indices. We then investigate their application to cluster analysis in data streams. While the results of any online clustering algorithm can be analyzed using the proposed iCVIs, we focus on two clustering algorithms - a crisp clustering algorithm from the general data stream clustering category (type 2a), viz., sk-means; and the *online elliptical clustering* (OEC) clustering algorithm from the time-series clustering category (type 2b), which is a soft/fuzzy clustering algorithm.

4 Problem Statement and Definitions

Traditional batch clustering algorithms aim to find crisp or fuzzy/probabilistic k-partitions of a collection of n samples of *static* data, viz., $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\} \subset \Re^p$. All vectors in this article are column vectors. Crisp and fuzzy partitions of X are conveniently represented by matrices in the following sets:

$$\begin{aligned} M_{fkn} = \{\, U \in \Re^{kn} : \textit{for } 1 \le i \le k, 1 \le j \le n : \\ 0 \le u_{ij} \le 1 : \sum_{i=1}^{k} u_{ij} = 1 \;\forall j;\; \sum_{j=1}^{n} u_{ij} > 0 \;\forall i \,\}\,; \end{aligned} \tag{1}$$

$$M_{hkn} = \{U \in M_{fkn} : u_{ij} \in \{0, 1\}\, \forall i, j\}\,. \tag{2}$$

Now suppose that n inputs have arrived sequentially in the streaming data, and we have found, from these n inputs $U_n \in M_{fkn}$, a set of soft/fuzzy clusters of X, together with a set $V_n = \{\boldsymbol{v}_{1n}, \ldots, \boldsymbol{v}_{kn}\} \subset \Re^{kp}$ of cluster centers. We can use (U_n, V_n) to calculate various CVIs.

When input $\boldsymbol{x}_{n+1}$ arrives, it is used by an online clustering algorithm to find

the membership, $u_{i,n+1}$, of the new point in the i^{th} cluster, $1 \leq i \leq k$. Let the vector $\boldsymbol{u}_{n+1} = \{u_{i,n+1} | i = 1, \ldots, k\}$ be the label vector of $\boldsymbol{x}_{n+1}$ in the set of (k) clusters. The clustering algorithm will also use $\boldsymbol{x}_{n+1}$ to update $V_n \rightarrow V_{n+1}$.

Given the new input $\boldsymbol{x}_{n+1}$, its cluster assignment $\boldsymbol{u}_{n+1}$ and the updated cluster centers V_{n+1}, we will derive one-step update formula for a particular CVI.

The calculation of $\boldsymbol{u}_{n+1}$ and updates to V are done using specific incremental clustering algorithms such as sk-means[3] or OEC.[5] The question posed here is how the value of the chosen CVI changes incrementally with this update. Fig. 1 illustrates the overall process. The two time series at the top of the figure form the input to the online clustering algorithm on the bottom right. When $\boldsymbol{x}_{n+1}$ becomes available, the online clustering algorithm finds the membership values $\boldsymbol{u}_{n+1}$, and updates the cluster centers to produce V_{n+1}. Then, $\boldsymbol{u}_{n+1}, V_{n+1}, \boldsymbol{x}_{n+1}$ are passed to the incremental cluster validation process. The objective in this paper is to answer the question "iCVI(n+1)=?" posed in the bottom left panel of Fig. 1.

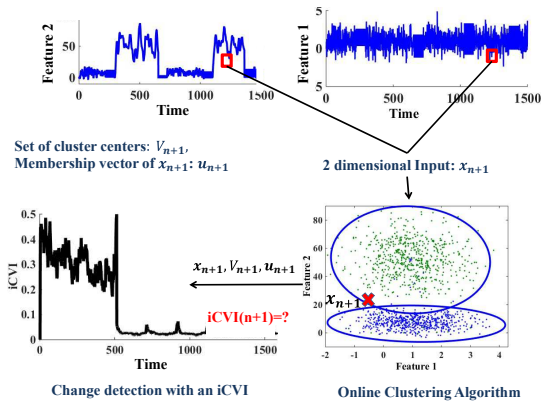


Figure 1: One-step update of incremental CVI in an online setting.

To answer this question, we first need to describe how $\boldsymbol{u}_{n+1}$ and V_{n+1} are calculated incrementally by online clustering algorithms. Section 5, briefly describes two algorithms to solve this problem.

5 Incremental Clustering Algorithms

The calculation of iCVIs depends on the information provided by a particular clustering algorithm. Therefore, we start by providing a brief overview of two different types of incremental clustering algorithms.

5.1 Sequential k-means

The sk-means algorithm shown below as Algorithm 1 and its variants have been studied in various forms in the literature of *self organizing maps* (SOMs).[3,19] Algorithm 1 records the basic sequential k-means method studied by Macqueen.[20]

Data: X - set of data points
Input: k - number of clusters
Note : $\|.\|$ is the Euclidean norm
Initialize V_k with the first k data points $V_k = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k\}$;
U_k is the $k \times k$ identity matrix;
Initialize the counter for each cluster $\{n_1, n_2, \ldots, n_k\}$ with 1;
foreach $\boldsymbol{x}_n$ *in the stream* **do**
  $m = \text{argmin}_{m \in \{1 \ldots k\}} \|\boldsymbol{x}_n - \boldsymbol{v}_m\|$;
  $n_m = n_m + 1$;
  $\boldsymbol{v}_m = \boldsymbol{v}_m + (\boldsymbol{x}_n - \boldsymbol{v}_m) / n_m$;
  $\boldsymbol{u}_n = (0, 0, \ldots, 1, \ldots, 0)^T \in \{0, 1\}^k$, where the 1 appears in the $m - th$ place, is appended to the current U as its $n - th$ column;
end

Algorithm 1: The basic sequential k-means algorithm (Macqueen[20]).

Macqueen's algorithm requires pre-specification of k, the number of clusters it builds. After initializing the k clusters by designating the first k points in X as the initial cluster centers, when the next input arrives, sk-means computes the distance from the next input to the k cluster centers, and assigns the input point to the cluster of the nearest prototype. Then the winning prototype is updated as shown in the third line of the *for* part of Algorithm 1. So at step $n + 1$ of the algorithm the k vector $\boldsymbol{u}_{n+1}$ has only one element with the value of 1 (the winner of a nearest prototype competition), and the remaining $k - 1$ values are 0.

5.2 Online Ellipsoidal Clustering

This section briefly reviews the *online ellipsoidal clustering* (OEC) algorithm developed in[5] as a fast online clustering algorithm for low dimensional data. This clustering algorithm is specific to time-series and does *not* require the number of clusters to be pre-specified. This algorithm has ellipsoidal cluster prototypes defined by k sets of means and covariance matrices $\left\{(\boldsymbol{m}_{i,n}, S_{i,n}^{-1})|i = 1, \ldots, k\right\}$ at any step n. Two different radii are considered for each ellipsoid, one called the *effective boundary*, i.e., the smaller radius, and the other one is called the *outlier boundary*. The outlier boundary prevents the cluster prototype from being affected by large outliers. These boundary thresholds are selected from the inverse of the chi-squared distribution, $(\chi^2)_p^{-1}(\gamma)$, where γ is the probability that a cluster member falls inside the ellipsoid. In the OEC clustering algorithm, cluster center $\boldsymbol{v}_{i,n+1}$ is formed from the sample mean $\boldsymbol{m}_{i,n+1}$ of each cluster, and $\boldsymbol{u}_{n+1}$ is formed by the values obtained from the soft assignments of the input point to the clusters, $\boldsymbol{u}_{i,n+1},\ 1 \leq i \leq k$, defined as

$$\boldsymbol{u}_{i,n+1} = \left[\sum_{j=1}^{k} \left(F_{i,n+1}/F_{j,n+1}\right)^{2/(m-1)}\right]^{-1}, \quad m \in (1, \infty), \tag{3}$$

where $F_{i,n+1} = (\boldsymbol{x}_{n+1} - \boldsymbol{m}_{i,n})^T S_{i,n}^{-1}(\boldsymbol{x}_{n+1} - \boldsymbol{m}_{i,n})$. Readers may recognize this as the membership update formula required by fuzzy k-means using the sample-based Mahalanobis norm for the inner product induced distance.[21] The OEC algorithm uses two other parameters that we need to set during the experiments.

Stabilization period (n_s) - An ellipsoidal prototype in $\Re^p$ is created by $p + 1$ consecutive distinct points, but to obtain a reliable estimate of $(\boldsymbol{m}_{i,n}, S_{i,n}^{-1})$, more data from the input stream is required. The integer n_s stabilizes these incremental estimates by temporarily disabling the OEC guard zone and new cluster detection tests until the current cluster contains n_s points.

Forgetting factor (λ_{OEC})- OEC has a special λ-prototype. When the λ-prototype does not overlap with any of the existing clusters in the system, a new cluster is formed. This prototype uses an exponential forgetting factor (λ_{OEC}) for this

purpose.

Now that we have described how $\boldsymbol{v}_{n+1}$ and $\boldsymbol{u}_{n+1}$ can be updated using two different clustering algorithms, we can describe how iCVIs use this information to assess the evolving clusters incrementally. In Section 6, we introduce an incremental calculation of a common cohesion measure in CVIs, then we discuss how two particular CVIs which use this cohesion measure can be updated using its previous value at time n with the additional information provided by $\boldsymbol{u}_{n+1}, V_{n+1}$ and $\boldsymbol{x}_{n+1}$.

6 The Incremental CVIs

Normally there is no external ground truth information available with streaming inputs, hence we explore the usage of internal CVIs to assess the evolving performance of online clustering algorithms that generate both memberships and cluster prototypes. We consider two well-known indices in this group that work for both hard and soft partitions.

The first index of this type is the *Xie-Beni* (XB) index.[7] A general formulation of this index for a batch collection of n inputs is given below in (4), where $\mathscr{A}$ is a positive-definite weight matrix which induces the inner product norm $\|x\|^2_{\mathscr{A}} = x^T \mathscr{A} x$:

$$XB_{m\mathscr{A}}(U, V; X) = \frac{J_{m\mathscr{A}}(U, V; X)}{n\left(\min_{i \neq j}\left\{\|\boldsymbol{v}_i - \boldsymbol{v}_j\|^2_{\mathscr{A}}\right\}\right)}$$

$$\text{, where } m \in [1, \infty), \text{ and} \tag{4}$$

$$J_{m\mathscr{A}}(U, V; X) = \sum_{j=1}^{n}\sum_{i=1}^{k}(u_{ij})^m \|\boldsymbol{x}_j - \boldsymbol{v}_i\|^2_{\mathscr{A}}.$$

The parameter m is called the fuzzifier of the model. For simplicity, we consider only $\mathscr{A} = I_p$ (the Euclidean norm) and $m = 2$, and drop subscripts, writing J_{2I_p} as J in the sequel.

The second index is a relative of the DB index[8] introduced by Araki *et al.*,[22]

$$DB(U,V;X) = \frac{1}{k}\sum_{i=1}^{k}\max_{j,j\neq i}\frac{L_i + L_j}{\|\boldsymbol{v}_i - \boldsymbol{v}_j\|^2}$$
$$L_i = \frac{\sum_{j=1}^{n} u_{ij}^2 \|\boldsymbol{x}_j - \boldsymbol{v}_i\|^2}{\sum_{j=1}^{n} u_{ij}^2}. \tag{5}$$

This is not a true generalization of the DB index because square roots are missing for this choice of $p = q = 2$ in .[8] but the values of (5) are closely related to the true DB index in the crisp case. The common part of these two (and many other) indices is the way that they capture the within-cluster dispersion. We define the *fuzzy within cluster dispersion*, for $U \in M_{fkn}$, as,

$$C_{i,n} = \sum_{j=1}^{n} (u_{ij})^2 \|\boldsymbol{x}_j - \boldsymbol{v}_{i,n}\|^2. \tag{6}$$

This is the only part of either index that depends directly on the input data. Therefore, we first develop an incremental calculation of this part of the index, and then use it to propose formulae for incremental calculation of these two indices.

6.1 Incremental Cluster Dispersion Measure

In this section, we derive a formula for (6) at time step $n+1$ based on its value at time step n. We assume that at time $n+1$, all of the previous input values $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$ have been discarded, so the only data point we have to work with is $\boldsymbol{x}_{n+1}$.The goal is to write the update formula in terms of the value in the previous step and a change to this value on seeing the $n+1^{st}$ input, i.e.,

$C_{i,n+1} = C_{i,n} + \Delta C_{i,n}$. We begin with

$$C_{i,n+1} = \sum_{j=1}^{n+1} (u_{ij})^2 \left\| \boldsymbol{x}_j - \boldsymbol{v}_{i,n+1} \right\|^2 \tag{7}$$

First we isolate the effect of the last point in the summation to obtain

$$C_{i,n+1} = \sum_{j=1}^{n} (u_{ij})^2 \left\| \boldsymbol{x}_j - \boldsymbol{v}_{i,n+1} \right\|^2 + \underbrace{(u_{i,n+1})^2 \left\| \boldsymbol{x}_{n+1} - \boldsymbol{v}_{i,n+1} \right\|^2}_{A_{i,n+1}}. \tag{8}$$

At this point if we could assume that $\|\boldsymbol{v}_{i,n+1} - \boldsymbol{v}_{i,n}\| \approx 0$, we would have $C_{i,n+1} = C_{i,n} + A_{i,n+1}$. Here, we are looking for an exact calculation of $C_{i,n+1}$ so we need to compute the effect of the change in the cluster centers. To isolate the change of the centers, we add and subtract $\boldsymbol{v}_{i,n}$ inside the Euclidean norm in the first term in (8), and rewrite the norm in terms of the (Euclidean) inner product,

$$C_{i,n+1} = \sum_{j=1}^{n} (u_{ij})^2 \left\langle \boldsymbol{x}_j - \boldsymbol{v}_{i,n} + \boldsymbol{v}_{i,n} - \boldsymbol{v}_{i,n+1}, \boldsymbol{x}_j - \boldsymbol{v}_{i,n} + \boldsymbol{v}_{i,n} - \boldsymbol{v}_{i,n+1} \right\rangle + A_{i,n+1}. \tag{9}$$

After few steps and some simplification we obtain

$$C_{i,n+1} = C_{i,n} + \overbrace{A_{i,n+1} + M_{i,n} B_{i,n+1} + 2Q_{i,n+1}}^{\Delta C_{i,n}} \tag{10}$$

where

$$Q_{i,n+1} = \sum_{j=1}^{n} (u_{ij})^2 \left\langle \boldsymbol{x}_j - \boldsymbol{v}_{i,n}, \boldsymbol{v}_{i,n} - \boldsymbol{v}_{i,n+1} \right\rangle \tag{11}$$

$$B_{i,n+1} = \|\boldsymbol{v}_{i,n} - \boldsymbol{v}_{i,n+1}\|^2 \tag{12}$$

$$M_{i,n+1} = M_{i,n} + u_{i,n+1}^2 : \quad M_{i,1} = u_{i,1}^2 \tag{13}$$

The term $Q_{i,n+1}$ in equation (11) depends on the previous (discarded) values of $\boldsymbol{x}_j$ and u_{ij} where $j = 1, \ldots, n$, so we cannot yet make the incremental calculation of $\Delta C_{i,n}$. Since the second part of the dot product in (11), i.e., $(\boldsymbol{v}_{i,n} - \boldsymbol{v}_{i,n+1})$, does not depend on j, we can write $Q_{i,n+1}$ as

$$Q_{i,n+1} = \left[\boldsymbol{v}_{i,n} - \boldsymbol{v}_{i,n+1}\right]^T \overbrace{\left[\sum_{j=1}^{n} \left(u_{ij}\right)^2 \left(\boldsymbol{x}_j - \boldsymbol{v}_{i,n}\right)\right]}^{G_{i,n}} \tag{14}$$

Using the same trick of adding and subtracting $\boldsymbol{v}_{i,n}$ in (14) we can write an incremental update formula for $G_{i,n+1}$,

$$\begin{aligned} &G_{i,n+1} = G_{i,n} + \Delta G_{i,n}, G_{i,1} = \overrightarrow{0} \\ &\Delta G_{i,n} = M_{i,n}(\boldsymbol{v}_{i,n} - \boldsymbol{v}_{i,n+1}) + \left(u_{i,n+1}\right)^2 \left(\boldsymbol{x}_{n+1} - \boldsymbol{v}_{i,n+1}\right). \end{aligned} \tag{15}$$

We now have all the components needed to calculate $\Delta C_{i,n}$. The two terms $A_{i,n+1}$ and $B_{i,n+1}$ are calculated directly and $Q_{i,n+1}$ and $M_{i,n+1}$ are incrementally calculated. The Algorithm 2 depicts a function that incrementally calculates the compactness.

As $n \to \infty$ the effect of each new sample on the total value of $C_{i,n}$ is expected to become small. In data streaming applications, which have, in theory, an infinite data stream, we are interested in how well a clustering algorithm keeps

Data: $\boldsymbol{v}_{i,n}, \boldsymbol{v}_{i,n+1}, u_{i,n+1}, \boldsymbol{x}_{n+1}$
Input : $G_{i,n}, M_{i,n}, C_{i,n}$
Output: $G_{i,n+1}, M_{i,n+1}, C_{i,n+1}$
`/* note` $i = 1, \ldots, k$
foreach $i \in \{1, \ldots, k\}$ **do**

$$Q_{i,n+1} = [\boldsymbol{v}_{i,n} - \boldsymbol{v}_{i,n+1}]^T G_{i,n};$$
$$B_{i,n+1} = \|\boldsymbol{v}_{i,n} - \boldsymbol{v}_{i,n+1}\|^2;$$
$$A_{i,n+1} = (u_{i,n+1})^2 \|\boldsymbol{x}_{n+1} - \boldsymbol{v}_{i,n+1}\|^2;$$
$$C_{i,n+1} = C_{i,n} + A_{i,n+1} + M_{i,n}B_{i,n+1} + 2Q_{i,n+1};$$
$$G_{i,n+1} = G_{i,n} + M_{i,n}(\boldsymbol{v}_{i,n} - \boldsymbol{v}_{i,n+1}) + (u_{i,n+1})^2 (\boldsymbol{x}_{n+1} - \boldsymbol{v}_{i,n+1});$$
$$M_{i,n+1} = M_{i,n} + u_{i,n+1}^2;$$

end

Algorithm 2: The incremental compactness function for data point $\boldsymbol{x}_{n+1}$.

up with the evolution of cluster structure of points in the stream. Our objective is to use iCVIs to capture the quality of the partitions over a window of the most recent observations at any point in time.

Exponential fading memory is a common approach in online learning methods. In this approach a forgetting factor $0 < \lambda < 1$ is included in the incremental estimations so that the data sample from f steps before the current sample is weighted by λ^f. In this way, older samples become less and less relevant to the current estimation. The batch representation of $C_{i,n}$ with a forgetting is shown in (16).

$$C_{\lambda i,n} = \sum_{j=1}^{n} \lambda^{n-j} u_{ij}^2 \|\boldsymbol{x}_j - \boldsymbol{v}_{i,n}\|^2 \tag{16}$$

An argument similar to the one used to derive equation (10) leads to an incremental update formula for $C_{\lambda i,n+1}$.

$$Q_{\lambda i,n+1} = (\boldsymbol{v}_{i,n} - \boldsymbol{v}_{i,n+1})\, G_{\lambda i,n} \tag{17}$$

$$B_{i,n+1} = \|\boldsymbol{v}_{i,n} - \boldsymbol{v}_{i,n+1}\|^2 \tag{18}$$

$$A_{i,n+1} = u_{i,n+1}^2 \|\boldsymbol{x}_{n+1} - \boldsymbol{v}_{i,n+1}\|^2 \tag{19}$$

$$\begin{aligned} C_{\lambda i,n+1} &= \lambda C_{\lambda i,n} + \Delta C_{\lambda i,n} \\ \Delta C_{\lambda i,n} &= 2\lambda Q_{\lambda i,n+1} + \lambda M_{\lambda i,n} B_{i,n+1} + A_{i,n+1} \end{aligned} \tag{20}$$

$$\begin{aligned} G_{\lambda i,n+1} &= \lambda G_{\lambda i,n} + \Delta G_{\lambda i,n} \\ \Delta G_{\lambda i,n} &= \lambda M_{\lambda i,n}(\boldsymbol{v}_{i,n} - \boldsymbol{v}_{i,n+1}) + (u_{i,n+1})^2 (\boldsymbol{x}_{n+1} - \boldsymbol{v}_{i,n+1}) . \end{aligned} \tag{21}$$

$$M_{\lambda i,n+1} = \lambda M_{\lambda i,n} + u_{i,n}^2 : \quad M_{\lambda i,1} = u_{i,1}^2 \tag{22}$$

In the next section, we use these formulas to derive two incremental versions of the XB and DB indices at equations (4) and (5).

6.2 Incremental Xie-Beni Index

Let XB($n+1$) denote the value of the XB index we seek when $\boldsymbol{x}_{n+1}$ arrives after n inputs have been processed. Let J_{n+1} denote the value of J at step $n+1$. To compute XB incrementally, i.e., to compute XB($n+1$), we need an incremental update for J_{n+1} and for the denominator of (4). The numerator of XB is updated using the value of $C_{i,n+1}$ from Algorithm 2, and the denominator is calculated with the updated centers V_{n+1}. Equation (23) shows the one step

update of J_{n+1} at step $n+1$.

$$J_{n+1} = \sum_{i=1}^{k} C_{i,n+1}. \tag{23}$$

Let

$$h_{n+1} = \min_{i \neq j, \boldsymbol{v}_i, \boldsymbol{v}_j \in V_{n+1}} \left\{ \|\boldsymbol{v}_i - \boldsymbol{v}_j\|^2 \right\}. \tag{24}$$

Then value of the incremental XB index, is

$$XB(n+1) = \frac{J_{n+1}}{(n+1)h_{n+1}}. \tag{25}$$

For batch clustering, the case $k = 1$ is usually not considered. However, in the streaming environment and in algorithms like OEC, the number of clusters dynamically changes and starts from $k = 1$. With one cluster, h_{n+1} is undefined in (24). When $k = 1$, we replace (24) with $h_{n+1} = \max\left\{h_n, \|\boldsymbol{v}_1 - \boldsymbol{x}_{n+1}\|^2\right\}$.

The one step update of iXB with forgetting is obtained by replacing $C_{i,n+1}$ with $C_{\lambda i,n+1}$ in (23) to obtain in $J_{\lambda,n+1}$,

$$XB_\lambda(n+1) = \frac{(1-\lambda)J_{\lambda,n+1}}{h_{n+1}}. \tag{26}$$

Please note that XB$(n+1)$ and XB$_\lambda(n+1)$ are the values of the incremental XB indices without and with the forgetting factor after $\boldsymbol{x}_{n+1}$ is processed, while iXB and iXB$_\lambda$ are the names of the incremental XB models.

6.3 Incremental DB Index

Let DB(n) denote the value of the Davies-Bouldin index after n inputs. We want to compute incrementally updated values DB$(n+1)$ and DB$_\lambda(n+1)$ when input $\boldsymbol{x}_{n+1}$ arrives. We need to normalize $C_{i,n+1}$ and $C_{\lambda i,n+1}$ with the number of data points in the i^{th} cluster. The index DB$(n+1)$ at time step $n+1$ can

be written as

$$DB(n+1) = \frac{1}{k}\sum_{i=1}^{k}\max_{j,j\neq i}\frac{L_{i,n+1}+L_{j,n+1}}{\|\boldsymbol{v}_{i,n+1}-\boldsymbol{v}_{j,n+1}\|^2}, \tag{27}$$

where

$$L_{i,n+1} = \frac{C_{i,n+1}}{M_{i,n+1}}. \tag{28}$$

To calculate the index with the forgetting factor, $DB_\lambda(n+1)$, we need to define $L_{\lambda i,n+1}$ to be used instead of $L_{i,n+1}$ and we control the decay of the denominator by clamping the decay at 1 with max function.

$$L_{\lambda i,n+1} = \frac{C_{\lambda i,n+1}}{\max\{1, M_{\lambda i,n+1}\}}. \tag{29}$$

Table 1: Summary characteristics of the datasets used in the evaluations.

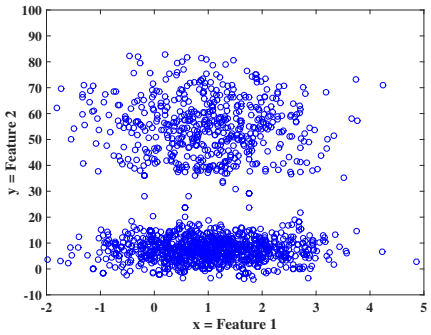
Dataset	# samples (n)	# Lbls (k)	# dim (p)	Noise Attributes	Labeling
S1	1955	2	2	No Noise - Drift between Clusters	Exact
S2	2727	11	2	1% - Uniform Random	Exact
S3	2000	10	2	1% - Systematic Random	Exact
LG	2016	3*	2	Unknown	Our guess*
GSA	9969	3*	8	Unknown	Our guess*

* These are unlabeled data sets, so we use physical arguments to justify the approximate number of clusters shown in column 2 for these two data sets.

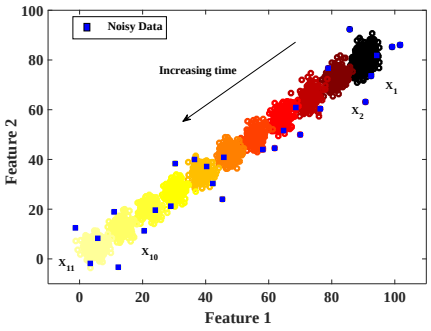
As with the XB case, DB(n) and DB($n+1$) are values of the incremental indices; iDB and iDB$_\lambda$ are the incremental models that produce them.

7 Computational Protocols

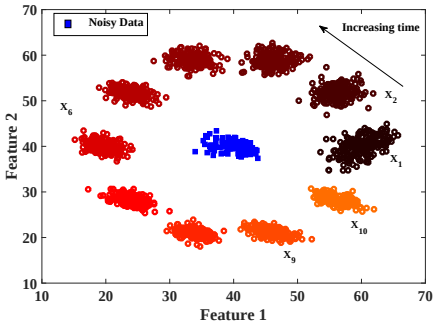
In this section, we first describe the synthetic and real-life datasets used in our evaluations and then we study the iCVIs iXB, XB$_\lambda(n)$, iDB and DB$_\lambda(n)$.



(a) S1: Synthetic Locally Linear Processes



(b) S2: Shifting Gaussian Distribution



(c) S3: Circular Shifting Gaussian Distribution with Systematic Noise

Figure 2: Scatter plots of three synthetic datasets. S2 and S3 show the progression of time with colors (depicted by an arrow), starting with black and becoming lighter with time.

7.1 Datasets and Parameters

The synthetic dataset $S1$ consists of two-dimensional vectors $(x_q, y_q), q > 2$, which are generated using two modes, M_1 and M_2, with different dynamic functions and input signals $(x_n, q = 3, 4, \ldots)$. Values of the independent variable (x) are random i.i.d. samples from a Gaussian distribution with $\mu = \sigma = 1$. Values of the dependent variable (y) from M_1 and M_2 are then computed according to (30) or (31) respectively.

$$\begin{aligned} y_n &= 1.018x_{n-1} + 1.801y_{n-1} - 0.8187y_{n-2} \\ y_0 &= y_1 = y_2 = 0 \end{aligned} \tag{30}$$

$$
\begin{aligned}
y_n &= x_{n-1} + 0.5x_{n-2} + 1.5y_{n-1} - 0.7y_{n-2} \\
y_0 &= y_1 = y_2 = 0
\end{aligned}
\tag{31}
$$

To build $S1$, we considered 4 mode changes between the two modes at uniform random intervals between 200 and 500 samples starting with M_1. Instead of a sudden shift between the modes, we gradually change the individual parameters of one mode to the other mode in 5 equal steps, during which we generate 10 samples in each intermediate mode. Fig. 2(a) shows a scatter plot of the $S1$ dataset with the input (Feature 1 = x) and output (Feature 2 = y).

The second synthetic dataset $S2$ (shown in Fig. 2(b)), is generated by considering two modes, M_1 and M_2, with different two-dimensional normal distributions $N(\mu_1, \Sigma_1)$ and $N(\mu_2, \Sigma_2)$ and 9 intermediate modes. The parameter values of the modes M_1 and M_2 are: $\Sigma_1 = \begin{pmatrix} 3.8418 & -2.6474 \\ -2.6474 & 4.8478 \end{pmatrix}$, $\Sigma_2 = \begin{pmatrix} 1.5239 & -0.5390 \\ -0.5390 & 1.6467 \end{pmatrix}$, and $\mu_1 = (95, 75)$ and $\mu_2 = (5, 5)$. M_1 is the initial mode, and M_2 is the final mode. M_1 is transformed as follows. First, $X_1 = 500$ samples $\{k = 1, \ldots, 500\}$ are drawn from M_1. Sampling continues as each individual value of the covariance matrix and the mean are changed in 10 equal steps from their values in M_1 to those in M_2 creating subsets $X_1, X_2, \ldots, X_{11}$ as shown in Fig. 2(b). The points are submitted to the stream in the order of creation, i.e., all the points in X_1, then X_2, and so on. After the first step, 200 samples $\{n = 501, \ldots, 700\}$ are taken from the new normal distribution. After each new step 200 more samples are added to the dataset. The final step ends at mode M_2. The squares show 1% of the samples from each normal distribution, which are perturbed by uniform noise from $[-10,\ 10]$. A small level of noise is added to this dataset to investigate how the algorithms react to noise.

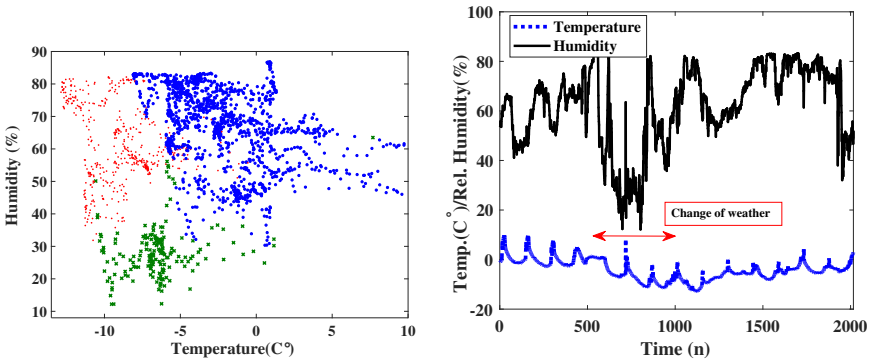
The third synthetic data set, S3, is generated by drawing samples from two-dimensional Gaussian distributions that rotate around a circle with 10 equal

shifts. Before each shift, 200 samples are generated using the current Gaussian. In this data set, the noise (the blue cluster in the center of Fig. 2(c)) is generated by a Gaussian at the center of the circle. At each of 10 steps a random number of samples between 1 and 20 are removed from the outer distribution and then this number of samples are drawn from and added to the inner noise distribution. This construction produces the 10 subsets $X_1, X_2, \ldots, X_{10}$ and a noise subset in the middle of the figure as shown in Fig. 2(c). Note that the center subset is not created in an ordered time sequence and its effects are observed as *systematic* noise in the stream. Table 1 summarizes the characteristics of the three synthetic datasets.

Column 2 of Table 1 specifies the number of physically labeled subsets in each data set. These subsets may, or may not, correspond to visually apparent or computationally acquired clusters. For example, if you imagine Fig. 2(b) without the colors (which show the labels and times), most observers would assert that this data set has only 1 cluster. These three synthetic data sets can be obtained by contacting the first author.

We also use two real-life datasets. The LG dataset is from a collection of weather station nodes in the *Le Genepi* (LG) region in Switzerland.[23] Two weeks of data at node 18 starting from October 10th 2007 are used in the evaluation. We use average surface temperature (T) and humidity (H) readings at node 18 over 10-minute intervals to form a total of 2016 two dimensional input vectors $\{x_n = (T_n, H_n)\}$.

Fig. 3(a) shows a scatter plot of the LG data. 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. Fig. 3(b) shows this change of weather in time-series plots of temperature and relative humidity data. Therefore, we assume that there are three physical events, and that these may correspond to 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 (blue),



(a) Scatterplot of LG

(b) Time-series view of the data

Figure 3: Scatterplot and time-series views of the LG data. The scatterplot shows three expected clusters.

day 7 in cluster 2 (green), and days 8 and 9 in cluster 3 (red).

The second real-life data set is a gas sensor array under dynamic gas mixtures from the UCI repository that contains conductivity samples at 100Hz obtained from 16 chemical sensors (4 unique sensors). The conductivity of these types of sensors changes in the presence of different gas mixture concentrations. More information on the generated data set can be found in Fonollosa *et al.*[24] We select two pairs of each unique sensor (8 sensors) and take the mean of their 100 samples per second over 5 minutes of the experiment as our evaluation data set. The resultant data contains 300 vectors with 8 features each. Each feature of one 8-vector is the mean of 100 samples in one second, so the input stream contains 300 points. We refer to this data set as GSA. During this five minutes the sensors are exposed to two different concentrations of gases CO and Ethylene. This will lead to three distinct behaviours in the dataset: no gas being present, presence of CO and presence of Ethylene.

7.2 Initialization

Macqueen's sk-means algorithm has only one parameter to choose, k. We use the recommended parameter values for OEC from.[5] These parameters are forgetting factor $\lambda = 0.9$, effective cluster boundary and outlier boundary threshold of

0.99 and 0.999 respectively, a stabilization period of $n_s = 20$, and and an OEC forgetting factor $\lambda_{OEC} = 0.9$.

The two clustering algorithms have slightly different initialization procedures. In sk-means the first k points are the initial cluster prototypes and the index calculations start at the $k + 1^{th}$ point. In the OEC clustering algorithm, the first $p + 1$ points are used to calculate a single cluster prototype and the cluster evaluation starts from point $p + 2$ with only one cluster in the system.

Upon start of the evaluation at step n, the iCVIs are initialized with $C_{i,n} = 0$, $M_{i,n} = n$ and $G_{i,n} = \overrightarrow{0}$ (zero vector in $\Re^p$). After initialization, the clustering algorithms and iCVIs process data one sample at a time.

8 Numerical Experiments

We first study the effect of forgetting in the indices. In these experiments we specify k to be the correct number of labels for sk-means for the synthetic datasets. We show that both indices with forgetting reveal more information about the data streams than when forgetting is not in use.

8.1 Forgetting or Not

In the iCVI indices (similar to their batch counterparts) each new data point at time n affects the overall value of the index with the weight $1/n$ (see (25)). Therefore, as n continues to increase, we expect $XB(n)$ and $DB(n)$ to become *saturated* by data points and lose the sensitivity they need to reflect changes due to new data inputs.

Fig. 4 shows the values of the two indices with and without forgetting for online clustering in $S2$ with the OEC algorithm. The times when the distribution of samples changes, which are known by construction of $S2$, specified as $n = 500, 700, 900, \ldots, 2500$, are indicated with red vertical lines. You can see a sudden jump in the indices at these times, and this exactly the point of monitoring the processing with an iCVI - to detect changes in the evolving structure

of the data. Even in a fairly small data set such as $S2$, the jumps in the indices with no forgetting as n increases (bottom of Fig. 4) becomes very small. A data distribution change occurs around 1750, but neither DB(n) nor XB(n) show this change. This suggests that these two indices are not as suitable for monitoring evolving clusters in streaming data as their forgetting factor counterparts, which are shown in the top part of Fig. 4. The jumps in the indices with forgetting factor are larger and much clearer than those in with no forgetting, and do not seem to depend on the cumulative number of samples processed by the index.

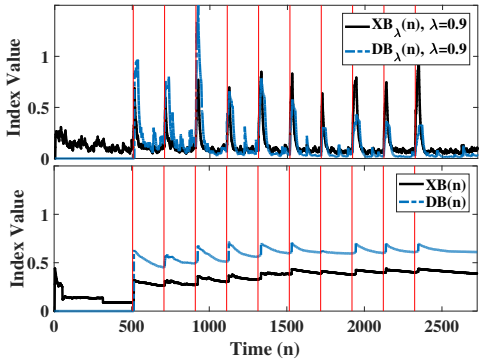
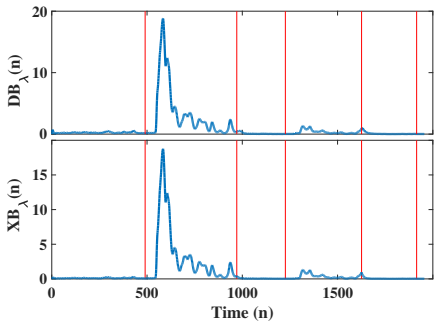


Figure 4: Values of XB(n), DB(n), XB$_\lambda$(n) and DB$_\lambda$(n) with $\lambda = 0.9$ in OEC processing of the S2 dataset.

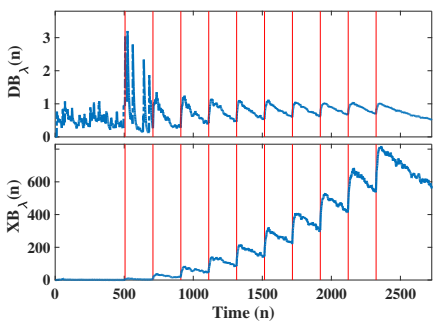
8.2 Interpretation of the iCVI model output over time

To study the usefulness of XB$_\lambda$(n) and DB$_\lambda$(n) in interpreting the results of online clustering algorithms, we first plot their values over time in clusters found by sk-means in the three synthetic datasets. In most cases because of the larger number of clusters and lower accuracy of sk-means in finding the expected clusters, the scale of the iCVI values for the sk-means approach is much higher than the values corresponding to OEC clusters. Therefore, we show the changes of XB$_\lambda$(n) and DB$_\lambda$(n) over time for only sk-means in Fig. 5. We will discuss the noteworthy trends in OEC using separate figures.

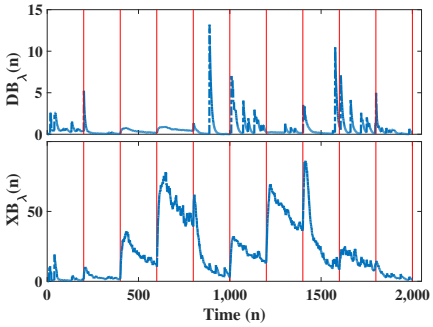
In Fig. 5, The indices XB$_\lambda$(n) and DB$_\lambda$(n) have almost identical performance for data set $S1$, as can be seen by comparing graphs of their values in Fig. 5(a).



(a) S1: Synthetic Locally Linear Processes



(b) S2: Shifting Gaussian Distribution



(c) S3: Circular Shifting Gaussian Distribution with Systematic Noise

Figure 5: Values of $\mathrm{XB}_\lambda(n)$ and $\mathrm{DB}_\lambda(n)$, $\lambda = 0.9$, for sk-means. The vertical lines (red) are times when a change occurs in the dataset.

But their performance is quite different for data sets $S2$ and $S3$, as can be seen in the top and bottom views in Figs. 5(b) and 5(c), respectively. We highlight two main traits in values of these indices. These traits relate to the appearance of new clusters in the stream (spikes in the index) and performance of the clustering algorithm in modeling the new cluster (the reduction of the index after a spike).

8.2.1 Appearance of new clusters

CVIs assess cohesion and separation so sudden changes in the value of a CVI indicate changes in the cohesion and separation of the clusters produced by the clustering algorithm. Thus, our expectation is that a sudden change in an online validity index will indicate the appearance of a new cluster in the data stream.

The points that result in the appearance of new clusters in the data stream change the cohesion of the clusters and cause spikes in the iCVI values.

The vertical lines in Fig. 5 mark the times where new clusters appear in the synthetic datasets. The sk-means algorithm performs reasonably well on the $S1$ dataset, completely fails for the $S2$ dataset (Fig. 6(a)), and partially identifies the evolving clusters in $S3$ (Fig. 6(b)). As we shall see, the OEC algorithm performs reasonably at detecting times when new distributions are created in all three datasets.

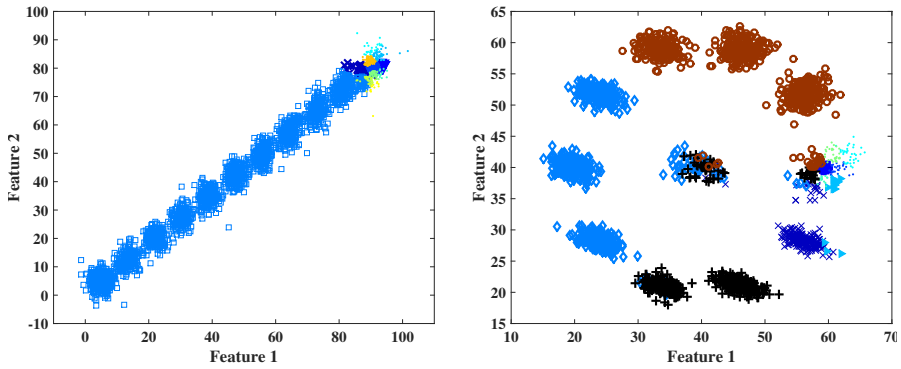
There is a jump in the values of both indices in Fig. 5 shortly after each new cluster is introduced in the data stream. As the algorithms learn the prototype that represents the new distribution in the data, the value of the index drops (bear in mind that these indices are all min-optimal). This behaviour is clearly shown in $\mathrm{XB}_\lambda(n)$ plots but the $\mathrm{DB}_\lambda(n)$ plot in the top view of Fig. 5(c) does not reflect this behaviour. We attribute this to partial identification of clusters in $S3$ and the systematic noise in this dataset which affects $\mathrm{DB}_\lambda(n)$ more than $\mathrm{XB}_\lambda(n)$.

8.2.2 Distress signals in the clustering algorithm

Another valuable asset of the graphs of incremental validity indices with forgetting is that a streaming plot of their values can exhibit signs of failure. The sk-means algorithm performs well in the $S1$ data set and we can see a sharp peak (sudden increase and decrease) in Fig. 5(a) after the first time the second cluster appears in the data. The XB_λ and DB_λ indices are almost identical in this data set. However, sk-means fails to identify the expected clusters in both $S2$ and $S3$ as shown by the end-state partitions in Fig. 6 even when the correct number of clusters are supplied to the algorithm. Let's see how the $\mathrm{XB}_\lambda(n)$ plots reflect this problem in the sk-means clustering algorithm.

Fig. 5(b) shows $\mathrm{XB}_\lambda(n)$ over time and the final clusters in $S2$ shown in Fig. 6(a) by sk-means. The $\mathrm{XB}_\lambda(n)$ plot shows that its values have an increasing trend. In this dataset, we have 11 clusters that appear one by one in the

stream. We know that the appearance of new clusters results in spikes in the $\mathrm{XB}_\lambda(n)$ values. After the appearance of a new cluster in the data, we expect the clustering algorithm to create a prototype for the cluster and update the partition to account for the newly observed cluster. Subsequently, this should bring any min-optimal *down* to similar (or lower) values that were observed before the new cluster appeared. The total failure of sk-means in $S2$ is evident by its inability to restore the $\mathrm{XB}_\lambda(n)$ values after a new cluster enters the data stream. Indeed, increasing values of these min-optimal indices signals that things are going awry! The $\mathrm{DB}_\lambda(n)$ values show smaller peaks and decreases but still shows that sk-means cannot find good clusters to sharply reduce the index. The graph in the top view of Fig. 4 shows the values of both indices for the OEC algorithm applied to $S2$, which identifies all the clusters correctly. The sharp peaks in the values verifies the fact that the clustering algorithm finds all the expected clusters.



(a) S2: Shifting Gaussian Distribution

(b) S3: Circular Shifting Gaussian Distribution with Systematic Noise

Figure 6: The terminal clusters produced by sk-means in $S2$ and $S3$

Figs. 5(c) and 6(b) tell a slightly different story for sk-means processing of $S3$. The sk-means clusters in this dataset provide some separation between the expected clusters. The separation is achieved by creating two subsets that each contain 3 of the original clusters, a subset with two of the presumptive clusters and the final cluster is identified correctly. At two points during the

experiment, there are significant reductions in the $\mathrm{XB}_\lambda(n)$ value (around 800 and 1500 samples). Fig. 6(b) shows that 8 of the 11 clusters in $S3$ are placed into three subsets (brown, blue (diamond) and black), and these three subsets also contain some of the points in the remaining 3 subsets. It is hard to see, but sk-means produces a total of 8 clusters (8 colors in Fig. 6(b)), none of which correspond to the labels of the data as constructed. So, sk-means does a poor job of finding the structure, but both iCVIs do a good job of reporting changes in the input stream that are perceived by sk-means.

The $\mathrm{DB}_\lambda(n)$ values in this dataset are very different from the $\mathrm{XB}_\lambda(n)$ values. The index $\mathrm{DB}_\lambda(n)$ generates large peaks around the noisy data and shows higher sensitivity to systematic noise in the data.

Sharp peaks corresponding to sudden drops in $\mathrm{XB}_\lambda(n)$ after the spikes (appearance of a new cluster) relates to the clustering algorithm appropriately creating a prototype for the cluster, while a gradual decrease after a spike is a sign of the failure of the algorithm to identify the new cluster. The $\mathrm{DB}_\lambda(n)$ index is more sensitive to large amplitude local noise and generates smaller peaks for the new clusters. This hinders the interpretation of the clustering results in terms of appearance of new clusters and learning problems in the algorithm.

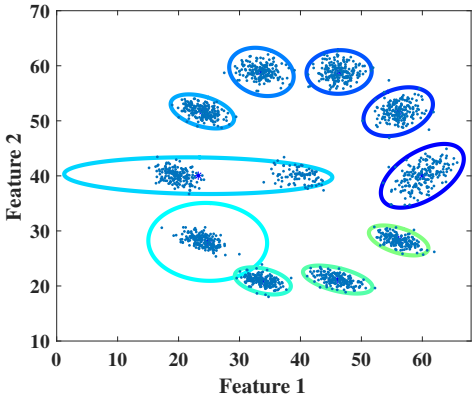


Figure 7: Terminal OEC clusters on $S3$ dataset.

So far we only looked at problems in the sk-means algorithm because OEC finds all the expected clusters in these three synthetic datasets. However, In

$S3$, as shown in Fig. 7, one of the cluster prototypes has been expanded and covers the systematic noise in the centre of the plot (The expanded prototype is a single prototype that represents the two clusters of data captured by the horizontally elongated ellipse in Fig. 7). Fig. 8 shows the values of the two indices for OEC clustering in this dataset. Both indices have similar trends with $\mathrm{DB}_\lambda(n)$ producing larger peaks for noisy data. Neither graph in Fig. 8 experiences a drop after creating this prototype. This indicates that OEC does not produce a good model of the data which matches the visual assessment of the clusters.

The overall conclusion from our synthetic data experiments is that $\mathrm{XB}_\lambda(n)$ is much more effective than $\mathrm{DB}_\lambda(n)$ in monitoring the performance of the clustering algorithms, so the remaining discussion will involve only $\mathrm{XB}_\lambda(n)$.

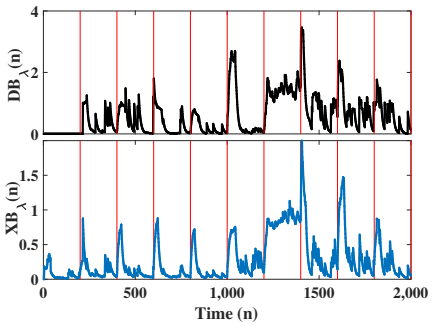


Figure 8: $\mathrm{XB}_\lambda(n)$ and $\mathrm{DB}_\lambda(n)$ values for OEC clusters in $S3$

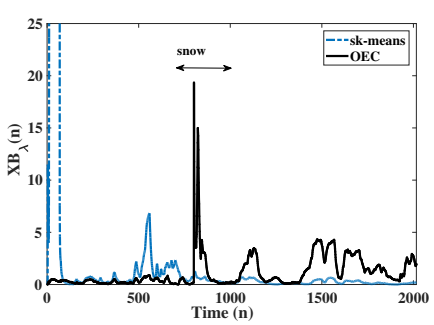
8.3 Experiments with real data

Since we have only assumed information about the ground truth in the real datasets it is harder to interpret the $\mathrm{XB}_\lambda(n)$ values. We will use the insights obtained from our analysis of the synthetic datasets to understand the behaviour of $\mathrm{XB}_\lambda(n)$ when clustering in the real-life datasets. Fig. 9(a) shows values of $\mathrm{XB}_\lambda(n)$ calculated for both OEC and sk-means (with $k = 3$ clusters) in the LG dataset. The major event in this dataset is a snowfall event (the approximate location of the event between about $n = 700$ and $n = 900$ is marked in the

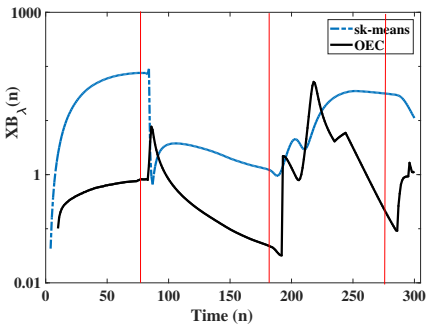
figure). OEC identifies three clusters in this dataset [5] which correspond to normal days, high wind before the snow and the snow. The two very close sharp spikes followed by drops in both indices values corresponds the spike for the wind before the snow and the snow fall as identified by OEC. The first spike corresponds to the time when the high wind had become the dominant feature in the area. Some level of wind can be observed from step 600.

The index values for sk-means clusters do not correspond to the major events in the dataset. There is only one major peak around the very start of the high winds at about step 600. Since sk-means does not consider the correlation in time between the data points, it fails to account for the temporal nature of the clusters in the LG data. As shown in Fig. 7.1, we must to look at the evolution of data in time to see the major events and the scatter plot of the data does not show a clear cluster tendency. The large overlap between clusters of data corresponding to major events leads to the fact that aligning clusters with the events does not guarantee the maximal separation. This is reflected in the index values for sk-means in Fig. 9(a) that shows no clear increasing trend or particularly slow reduction in the index values. However, the index values for sk-means clusters are mainly lower than those for OEC clusters showing that sk-means clustering creates relatively better clusters in terms of cohesion and separation.

The GSA dataset is collected in a more controlled environment than LG and hence, has a more recognizable cluster tendency. In this dataset two different gas concentrations are introduced during the experiment. When the gas is introduced (times indicated by the red vertical lines) there is a delay until the sensors react to the presence of the gas, which is seen as a delay of the values in Fig. 9(b). The peaks in both indices occur a few seconds after the introduction of the gas. For the first event at about $n = 75$, a gas is introduced when no other gas is present, while the second event at about $n = 180$ corresponds to removing one gas and introducing another gas. In the second case, the indices show two peaks close to each other for both clustering algorithms. The last event at about



(a) LG: Sensorscope project in Le Genepi



(b) GSA: Gas Sensor Array. Vertical lines indicate the times where concentration of gases change

Figure 9: The $\mathrm{XB}_\lambda(n)$ index over the real-life datasets. The y-axis of GSA plot is shown in logarithmic scale.

$n = 270$ corresponds to emptying the chamber and since the empty chamber had been seen by the clustering algorithms, this results in a much smaller peak in the graphs. OEC has smaller $\mathrm{XB}_\lambda(n)$ values than sk-means, which indicate that in this dataset, OEC results in a better separation between the expected clusters than sk-means.

9 Discussion and Conclusions

During the course of this study, we realized that several well-defined notions for batch clustering do not carry over to the streaming data case. The term "streaming clustering" is not an accurate descriptor of the algorithms used, because at the end of processing, there are no clusters, only a cluster footprint in the form of cluster centers, and in the case of OEC, cluster centers and covariance matrices. Moreover, the idea underlying cluster validity for candidate partitions of batch data does not apply to streaming data, because there are no partitions to evaluate. We believe that this paper describes a brand new method for monitoring the performance of streaming clustering algorithms. In turn, the incremental CVIs we have derived are also misnamed, because they are not really cluster validity indices: they are functions derived from batch

CVIs that track computational performance, enabling the user to control and analyze the dynamic performance of the streaming algorithms to which they are attached. A better term for our iCVIs might be something like *incremental Performance Monitors* (iPMs).

In this article, we introduced online iCVIs (or iPMs) and derived forgetting and non-forgetting versions of two well-known internal validity indices (the Xie-Beni indices (XB(n) and XB$_\lambda$(n)) and the Davies-Bouldin indices (DB(n) and DB$_\lambda$(n))). Our experiments used two different styles of algorithms for streaming clustering: sk-means, which does not account for the historical context of streaming data; and OEC, which retains a history of time dependency in clusters through the retention of its cluster statistics (means and covariances). When streaming clustering algorithms work well, we should observe a fairly flat iCVI. When a change occurs and new cluster appears in the stream, we should see a sudden jump in iCVI values followed by a similarly sudden drop. Jumps show the change and drops show whether the clustering algorithm has managed to create a prototype to account for the change. If the drop is not significant it shows that the clustering algorithm has not created a prototype for the change and instead assigns the incoming points to the previous cluster and updates the previous cluster prototype, resulting is in slight - but not significant - improvement in the iCVI value.

An increasing trend in any min-optimal index affords a means for sending distress signals about evolving clusters to real time monitors, because the increasing trend is counter to the mathematical property of min-optimality.

Another aspect of the iCVI approach worth mentioning is that incremental cluster validity measures lack the bias towards any specific type of cluster structure that their batch predesessors often exhibit. It is well known that many internal and external batch CVIs suffer from various bias problems when used to evaluate partitions obtained by batch processing. For example, Nguyen *et al.*[25] discuss methods that offset bias due to statistical chance in many external CVIs based on information-theoretic principles. Lie *et al.*[26] show that the

distribution of ground truth partitions can bias seven well known external measures such as mutual information and the Rand and Adjusted Rand indices. Some internal CVIs are monotonic in c (cf. discussions about normalization to counteract monotonicity of the partition coefficient and partition entropy in [21]). Finally, it is well known that some CVIs prefer a certain cluster shape. For example, the internal Davies-Bouldin index [8] is often thought to be biased towards a preference for spherical clusters. But this type of bias is usually due to using the Euclidean norm, which induces a spherical topology on the input space. This type of bias can usually be avoided by using a different metric.

The iCVIs in this paper, when used incrementally with forgetting factor, cannot exhibit any of the bias types mentioned in the previous paragraph, because there are no (global) shapes, or partition histories, that can affect the overall computation made by an iCVI. This is a subtle but important point that provides another reason to develop and study this new class of incremental measures.

Our experiments indicate that out of the four incremental indices {XB(n),XB$_\lambda$(n), DB(n),DB$_\lambda$(n)}, the most consistent index with respect to our belief about the evolving structure of the data seems to be XB$_\lambda$(n). And of the two clustering algorithms used, OEC seems to perform much better than sk-means. But OEC is a fast clustering algorithm that is most effective for low-dimensional data. A definitive conclusion about the utility of iCVIs requires many more tests. There are many, many internal CVIs. Our next focus will be on deriving other incremental validity indices and conducting comparative studies among different types of indices.

10 Acknowledgment

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

References

[1] S. Guha, A. Meyerson, N. Mishra, R. Motwani and L. O'Callaghan, *IEEE Transaction on Knowledge and Data Engineering*, 2003, **15**, 515–528.

[2] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. de Carvalho and J. ao Gama, *ACM Computing Surveys*, 2013, **46**, 13–31.

[3] M. Ackerman and S. Dasgupta, Neural Information Processing Systems (NIPS), Montreal, Canada, 2014.

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

[5] M. Moshtaghi, J. Bezdek and C. Leckie, Proceedings of SIAM Conference on Data Mining, Florida, USA, 2016.

[6] O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. Perez and I. Perona, *Pattern Recognition*, 2013, **46**, 243 – 256.

[7] X. L. Xie and G. Beni, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1991, **13**, 841–847.

[8] D. L. Davies and D. W. Bouldin, *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 1979, **1**, 224–227.

[9] C. C. Aggarwal, J. Han, J. Wang and P. S. Yu, in *On Clustering Massive Data Streams: A Summarization Paradigm*, Springer US, 2007, vol. 31, pp. 9–38.

[10] N. Ailon, R. Jaiswal and C. Monteleoni, Neural Information Processing Systems 2009, 2009, pp. 10–18.

[11] M. Salehi, C. Leckie, M. Moshtaghi and T. Vaithianathan, Proceedings of PAKDD, 2014, pp. 461–473.

[12] P. Angelov and X. Zhou, *IEEE Transactions on Fuzzy Systems*, 2008, **16**, 1462–1475.

[13] F. Cao, M. Ester, W. Qian and A. Zhou, SIAM Conf. on Data Mining, 2006, pp. 328–339.

[14] P. Kranen, I. Assent, C. Baldauf and T. Seid, *Knowledge and Information Systems*, 2011, **29**, 249–272.

[15] N. Mozafari, S. Hashemi and A. Hamzeh, *Artificial Intelligence Research*, 2014, **3**, 38–45.

[16] G. W. Milligan and M. C. Cooper, *Psychometrika*, 1985, **50**, 159–179.

[17] K. Schermelleh-Engel, M. Moosbrugger and H. Helfried, *Methods of Psychological Research*, 2015, **8**, 23 – 74.

[18] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Kluwer Academic Publishers, Norwell, MA, USA, 1981.

[19] N. R. Pal, J. C. Bezdek and R. J. Hathaway, *Neural Networks*, 1996, **9**, 787 – 796.

[20] J. MacQueen, In 5th Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281–297.

[21] J. C. Bezdek, *A Primer on Cluster Analysis: Four Basic Methods that (Usually) Work*, First Design Publishing, Sarasota, FL., USA, 2017.

[22] S. Araki, H. Nomura and N. Wakami, Proceedings of the Second IEEE International Conference on Fuzzy Systems, San Francisco, USA, 1993, pp. 719–724.

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

[24] J. Fonollosa, S. Sheik, R. Huerta and S. Marco, *Sensors and Actuators B: Chemical*, 2015, **215**, 618 – 629.

[25] N. X. Vinh, J. Epps and J. Bailey, *The Journal of Machine Learning Research*, 2010, **11**, 2837–2854.

[26] Y. Lei, J. C. Bezdek, S. Romano, N. X. Vinh, J. Chan and J. Bailey, *Pattern Recognition*, 2017, **65**, 58–70.