Evolving Systems
https://doi.org/10.1007/s12530-021-09408-y

ORIGINAL PAPER

EvolveCluster: an evolutionary clustering algorithm for streaming data

Christian Nordahl[1] · **Veselka Boeva**[1] · **Håkan Grahn**[1] · **Marie Persson Netz**[1]

Received: 3 March 2021 / Accepted: 27 October 2021

Abstract
Data has become an integral part of our society in the past years, arriving faster and in larger quantities than before. Traditional clustering algorithms rely on the availability of entire datasets to model them correctly and efficiently. Such requirements are not possible in the data stream clustering scenario, where data arrives and needs to be analyzed continuously. This paper proposes a novel evolutionary clustering algorithm, entitled EvolveCluster, capable of modeling evolving data streams. We compare EvolveCluster against two other evolutionary clustering algorithms, PivotBiCluster and Split-Merge Evolutionary Clustering, by conducting experiments on three different datasets. Furthermore, we perform additional experiments on EvolveCluster to further evaluate its capabilities on clustering evolving data streams. Our results show that EvolveCluster manages to capture evolving data stream behaviors and adapts accordingly.

Keywords Evolving data stream · Clustering · Data stream clustering

1 Introduction

In recent years, data has become an integral part of our daily lives. Due to advances in hardware infrastructures, there are endless possibilities available to collect any type of data at a rapid pace. Examples of streaming data sources include weather sensors, mobile applications, Instagram posts, electricity consumption, shopping records, etc. (Bifet et al. 2010b).

These data streams are endless information sources that arrive in a timely fashion. Incoming data tends to be unlabeled, as it requires too much effort to label it by hand. The immense amount of data proves to be hard to manage with traditional supervised machine learning algorithms and caused the emergence of unsupervised learning techniques. Unsupervised learning is a branch of machine learning where algorithms learn by themselves, identifying the underlying structure of a dataset. Depending on the application, the results from the unsupervised learning algorithms can be used directly for analysis or as an intermediary step to gain an understanding of the data. One of the branches of unsupervised learning is the task of clustering analysis.

Clustering is the process of grouping data instances into groups based on their similarity to each other (Gama 2010). Intuitively, instances within a cluster are more similar to each other than to other instances belonging to another cluster (Jain et al. 1999; Zubaroglu and Atalay 2021). The objective of clustering algorithms is to detect these underlying characteristics of the instances that make each cluster unique. Traditional clustering algorithms, such as *k*-means (Lloyd 1982), require the entire dataset to be available. In data stream clustering, the data arrives incrementally in such a high quantity and pace that traditional clustering methods cannot cope with (Gama 2010).

In the evolving data stream scenario, we have a continuous data stream that contains changes over time. These changes cause traditional offline models to become obsolete over time as the new data no longer conforms to how the model has been trained. Incremental clustering algorithms, such as the one introduced by (Lughofer 2008), are one way to address the problem of evolving data streams. These algorithms process elements on a step-wise basis and injects them into the existing clustering solution, updating the clusters by merging or splitting if needed.

Many applications do, however, not necessarily need such rapid adaptations given by incremental clustering algorithms. Instead, by approaching the data stream segmentally, it is possible to view how the data is changing over time directly. For example, electricity providers can identify if

✉ Christian Nordahl
christian.nordahl@bth.se

[1] Department of Computer Science, Blekinge Institute of Technology, Valhallavägen 1, 37141 Karlskrona, Sweden

Published online: 13 November 2021

and how the electricity consumption trend has altered in a single household, neighborhood, or an entire city, and determine if any remediation is required. Likewise, an online retailer can identify consumer shopping trends and how they change over time. When an overarching view of how the data aligns with previous structures and how it changes over time is desirable, there is a lesser need for direct updates to the clustering models.

In this study, we propose a novel evolutionary clustering algorithm capable of modeling data streams containing evolving data, entitled EvolveCluster, a continuation of our previous work (Boeva and Nordahl 2019). Instead of processing elements individually, we collect data over a defined period (creating segments) to trace how the data evolves. Two similar approaches have been identified to compare and evaluate the proposed algorithm, namely PivotBiCluster (Ailon et al. 2012) and Split-Merge Evolutionary Clustering (Boeva et al. 2019). Both these algorithms address the evolving data stream scenario by dividing the data into segments. In contrast to EvolveCluster, PivotBiCluster and Split-Merge Clustering map previous clustered data segments to fit with the newly arrived data segment. Both these algorithms combine the current data segment with the previous one by identifying similarities between the clusters from the two segments. However, the main drawback with both PivotBiCluster and Split-Merge Clustering is that they both require each data segment to be clustered in advance.

Our main contributions are as follows:

- We introduce a new algorithm, entitled EvolveCluster, that is especially targeted at evolving data streams. The design of the algorithm makes it easy to understand how trends and patterns appear in the data segments (Sect. 4.2).
- We provide a thorough analysis of the computational complexity of the proposed algorithm (Sect. 4.3).
- We evaluate the performance of EvolveCluster, PivotBiCluster, and Split-Merge Clustering, and we identify and discuss their strengths and weaknesses (Sects. 6 and 7).

2 Background

In this section we provide the necessary background information. First a description of clustering analysis is provided, with a specific definition of *k*-medoids. We continue by introducing concept drift, dissimilarity measures, and conclude this section with an explanation of evaluation and validation measures.

2.1 Clustering algorithms

Clustering algorithms are designed to identify an underlying structure of data and use the detected relationships within the structure to group the data points into distinct groups. These algorithms usually decide upon themselves how to divide the data into subgroups, an unsupervised approach to increase knowledge about the data. There are numerous ways of approaching this task and we can group them into five major categories: density-based, grid-based, hierarchical, model-based, and partitioning algorithms (Berkhin 2006). This study focuses on partitioning algorithms due to the proposed evolutionary clustering algorithms characteristics (see Sect. 4).

Partitioning algorithms differ from the other algorithm types in their need to define the number of clusters in advance. The number of clusters, usually denoted as k, is a parameter given to the algorithms when they are initialized. But, identifying an appropriate k in advance is not easy. A common approach to identify a suitable k is having the algorithm execute multiple times with an increasing k value. More sophisticated methods exist, where the data is analyzed in advance with an initialization algorithm that estimates how many clusters are present in the dataset (Arthur and Vassilvitskii 2006). These initialization algorithms, however, do not promise to produce an optimal solution.

One of the most prolific examples of a partitioning algorithm is the k-means algorithm. k-means starts by assigning k initial cluster centroids, either randomly or by an initialization algorithm. All data points are distributed into each cluster based on their distance to the centroids. The solution is refined by first electing a new cluster centroid, based on the mean values of each data object in the cluster, and then redistributing the data points accordingly. k-means refines the solution until changes are no longer made or until a maximum limit of iterations has been reached.

k-medoids, or Partitioning Around Medoids (Vinod 1969), is similar to k-means and generally seen as a sister algorithm. k-medoids, however, use actual data points as cluster centroids instead of creating synthetic centroids. This approach makes the algorithm more robust than k-means, being less susceptible to noise and outliers.

2.2 Concept drift

One of the phenomenons present in data streams, especially in evolving data streams, is how the data changes and evolves. This non-stationarity of data over time is referred to as concept drift (Khamassi et al. 2018). Depending on the data and how it changes over time, different types of concept drifts may exist in the streams. Wadewale and Desai (2015) divided concept drift in six categories: sudden, incremental, gradual, recurring, blip, and noise. Sudden concept drifts are abrupt changes to the data, while incremental and gradual drifts happen more slowly. A recurring drift is a sudden, gradual, or incremental drift that happens periodically. Blip and noise are defined as outliers and random instances that should be filtered out, respectively.

Concept drift is a crucial aspect of learning from evolving data streams. As the data evolves, the algorithm needs to be capable of adapting and continuously learn about the underlying data structure to model it correctly. In addition to our comparative experiments (Sects. 6.1–6.3), we perform an additional set of experiments to focus solely on EvolveCluster's ability to model data streams where concept drift is present (see Sect. 6.4).

2.3 Dissimilarity measures

Calculating the distance, or dissimilarity, between two objects is a requirement to enable the use of distance-based clustering algorithms such as k-medoids (Vinod 1969). These measures provide a numerical value that indicates how dissimilar or distant two data objects are. Numerous variants of measures exist, and their usage depends on the data itself. Two of the most common measures are the L_1 and L_2, commonly referred to as Manhattan and Euclidean distance (ED) (Wang et al. 2013), respectively. These two measures are relatively simple and tend to be very effective when the dataset has a lower dimensionality.

Based on the application, a variety of dissimilarity measures exist. Concerns such as dimensionality, computational efforts, type of data, etc., factor in choosing the measure (Shirkhorshidi et al. 2015). For instance, an electricity consumption dataset is a time series dataset. If the shape of the consumption, i.e., behavior, is desired, a measure such as DTW (Sakoe and Chiba 1978) is an eligible candidate measure. As an elastic measure, DTW could identify similar behaviors that occur at different times of day as closely related. If instead a strict measure was used, such as ED, those similar behaviors would likely not be identified as similar. However, if there was a concern that the behaviors should be performed at the exact time and place each day to be identified as similar, ED would be a better choice of measure Nordahl et al. (2019).

The datasets used in this study vary quite distinctively in their type and number of data points. None of them, however, is considered to be a high-dimensional dataset. Thus, we decided to use ED (Wang et al. 2013) on our datasets to focus on the algorithms and their properties. ED is defined as follows

$$\mathrm{ED}(q,p) = \sqrt{\sum_{i=0}^{n} \left(q_i - p_i\right)^2}, \tag{1}$$

where q and p are two data vectors consisting of n-dimensions and p_i and q_i are individual points in p and q, respectively.

2.4 Cluster validation measures

The data mining literature provides a wide range of different cluster validation measures, which are broadly divided into two major categories: *external* and *internal* (Jain and Dubes 1988). External validation measures have the benefit of providing an independent assessment of clustering quality, evaluating the clustering results with respect to a prespecified structure. Within the external evaluation, there are two distinct classes of measures: unary and binary (Handl et al. 2005). Unary measures often take a clustering solution as input and compare it against the ground truth. The clustering solution can be evaluated with regard to both the purity and the completeness of the clusters. F_1 is one example of such a validation measure (Chinchor 1992). In addition, to unary measures, a number of indices that assess the consensus between two partitioning solutions, based on the pairwise assignment of data points, are provided in the data mining litterature. Most of these indices are symmetric, making them well-suited for assessing the similarity of two clustering solutions, in which the Jaccard Index is a good example of (Jaccard 1912).

Internal validation techniques, on the other hand, avoid the need for using such additional knowledge. They evaluate the clustering solutions based upon the same information that were used to create the clusters, enabling them to evaluate the quality of the produced clustering solutions in different ways. Internal measures can be divided into four categories based on how they evaluate clustering solutions: *compactness*, *separation*, *connectedness*, and *stability* of the cluster partitions. A detailed overview of different types

of validation measures is available in (Halkidi et al. 2001; Vendramin et al. 2010).

Traditionally, many researchers working in data stream clustering apply well known validation measures to evaluate the clustering solutions produced by their data stream clustering algorithms (Silva et al. 2013), such as Sum of Squared Errors (SSE) and purity. Specific validation measures do, however, exist in the realm of data stream clustering. In 2011, Cluster Mapping Measure (CMM) is proposed as an effective measure for data stream clustering (Kremer et al. 2011). CMM is a combination score of missed objects, misplaced objects, and noise inclusion, and is based on the ground truth. More recently, several adaptations of well-known validation measures have been proposed, including Silhouette Index (Da Silva et al. 2020), Davies-Bouldin (Da Silva et al. 2020), and Xie-Beni (Moshtaghi et al. 2019). All of the aforementioned measures are, however, designed for incremental clustering algorithms. The algorithm proposed in this paper divides the stream into fixed-sized segments and separates the segments to analyze and evaluate them individually. Due to the algorithm's intended application and design, that a stream is divided into segments, it can be argued that it is not necessary to adopt the incremental validation measures. EvolveCluster does not process elements incrementally. Instead, each segment is statically clustered, which allows us to utilize traditional validation measures on each segment. Furthermore, as the algorithm only operates on entire segments and not individual instances, there is no directly applicable way to validate with these types of measures.

In the coming sub-sections, we describe and define the evaluation measures we apply in the study. We use two external (F_1-measure and Jaccard Index) and two internal (Silhouette Index and Average Intra-Cluster Distance) cluster validation measures to the clustering solutions generated by our experiments.

2.4.1 F_1-measure

F_1 is the harmonic mean of the precision and recall values of each cluster. Consider two clustering solutions, $A = \{A_1, \ldots, A_k\}$ and $B = \{B_1, \ldots, B_l\}$, of the same dataset. We define A as the known partitioning of the dataset and B as the partitioning produced by the applied clustering algorithm. We then define the F_1 for a cluster B_j as:

$$F_1(B_j) = \frac{2|A_i \cap B_j|}{|A_i| + |B_j|}, \tag{2}$$

where A_i is the cluster containing the maximum number of objects from B_j.

To evaluate the overall F_1 score for the clustering solution B, two common approaches are used, micro and macro average. Both versions are similar, but the macro average sees all classes as equal while the micro average corrects the score by each individual class's frequency. In this study, the datasets used (see Sect. 5.1.1) have a fairly even distribution of the corresponding classes. Therefore, the macro F_1 is used and for the clustering solution B it is defined as:

$$F_1(B) = \frac{1}{l} \sum_{j=1}^{l} F(B_j), \tag{3}$$

where l is the number of clusters within B. The F_1 score has a value between 0 and 1, with 1 indicating a perfect score.

2.4.2 Jaccard Index

For evaluating the stability of a clustering solution, Jaccard Index (JI) is a suitable candidate. Given two clustering solutions produced from the same dataset, A and B, we define JI between A and B as follows:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}, \tag{4}$$

where $|A \cap B|$ is the number of data points with the same class in the same clusters in A and B, and $|A \cup B|$ is the total number of data points in the same clusters in A and B. JI ranges from 0 to 1, where a higher value indicates a higher similarity between the clustering solutions.

2.4.3 Silhouette Index

Silhouette Index (SI) is a cluster validity index that is used to determine the quality of any clustering solution $C = \{C_1, \ldots, C_k\}$. It produces a score that is based on the compactness of each cluster and the separation between the clusters (Rousseeuw 1987). SI for a clustering solution C is defined as:

$$SI(C) = \frac{1}{m} \sum_{i=1}^{m} \frac{b_i - a_i}{max(a_i, b_i)}, \tag{5}$$

where a_i is the average distance from object i to the other objects in its cluster and b_i is the minimum average distance from i to the objects of the other clusters. SI ranges from -1 to 1, where a value closer to 1 indicates a better clustering solution, and a value on the negative side of the range

indicate that there are misplaced data points within the clustering solution.

2.4.4 Average Intra-Cluster Distance

Similarly to SI, the Average Intra-Cluster Distance (IC-av) measures how compact the produced clusters are. In contrast to SI, it does not assume a spherical shape of the produced clusters (Baya and Granitto 2013). Instead of calculating the radius around the clusters, IC-av produces a Minimum Spanning Tree (MST) of all data points based on the distance between the objects in the dataset. The edges containing the distance between the data points in the tree are then used to determine the compactness of the clusters in the clustering solution. For a particular clustering solution $C = \{C_1, \ldots, C_k\}$, IC-av is defined as:

$$\text{IC-av}\,(C) = \sum_{r=1}^{k} \frac{1}{n_r} \sum_{i,j \in C_r} d_{ij}^2, \tag{6}$$

where n_r is the number of objects in cluster C_r $(r = 1, 2, \ldots, k)$ and d_{ij} is maximum edge distance which represents the longest edge in the path joining objects i and j in the MST. IC-av produces a score between zero and the maximum value of the edges in the MST and should be minimized.

3 Related work

In this section, we provide a review of studies related to our work. At the end of the section, we specifically review the two algorithms PivotBiCluster and Split-Merge Evolutionary Clustering.

3.1 Evolving data streams

The data stream clustering scenario differs from traditional clustering because the data is usually not available in its entirety. Additionally, the data in the stream arrives at such a rapid pace and in large quantities that it is impossible to keep the data in the main memory (Gama 2010; Bifet et al. 2010b). Traditional algorithms, such as k-means (Lloyd 1982) and DBSCAN (Ester et al. 1996), rely on the entire dataset being present. A naive approach to apply traditional algorithms on data streams would be to re-cluster the entire solution at each increment. However, this approach is unfeasible both in regards to time constraints and the resources needed by the algorithms (Mousavi et al. 2015; Zubaroglu and Atalay 2021).

In addition to the quantity and rapidness of data in data streams, evolving data streams have the additional dynamics of non-stationarity data over time, also known as concept drift (Khamassi et al. 2018). Multiple approaches have been investigated to capture the dynamic aspects of evolving data streams, including (O'callaghan et al. 2002; Gama et al. 2011; Kriegel et al. 2011; Ghesmoune et al. 2015; Zhou et al. 2008; Lühr and Lazarescu 2009; Angelov and Zhou 2008). More specifically, Lughofer (2008) proposes an incremental algorithm, where each increment causes the affected cluster to be split and merged separately, which was further developed in (Lughofer 2012). Similarly, (Aaron et al. 2014) extends the k-means algorithm to a dynamic incremental clustering algorithm. A common idea for capturing evolving data streams' dynamic nature is to use incremental algorithms and add functionality to modify clusters by splitting and merging (Aaron et al. 2014; Lühr and Lazarescu 2009). In general, these algorithms are divided into two components: Online and Offline (Zubaroglu and Atalay 2021). The online component of the algorithms produces micro clusters that stay up to date which each increment of data objects that arrives, and the offline component runs periodically to finalize the clustering solution based on the produced microclusters.

3.2 Window based models

Generally, within data stream clustering, especially in contrast to traditional clustering, it can be more efficient to focus on the recent data instead of the entire stream. Several window models exist, but the following three are the most popular: damped window, sliding window, and landmark window (Zubaroglu and Atalay 2021). The damped window models approach the data limitation by incorporating a weight factor when the data is processed. No object is removed from the window, but older the data objects have lower importance for the model. It is usually performed by a negative exponential function, such as $f(t) = 2^{-\lambda t}$. SNCStream (Barddal et al. 2015) and its extension SNCStream+ (Barddal et al. 2016) operate in a damped window scenario in a single pass manner. They are based on social network theory and use homophily to identify non-hyper spherical clusters. Similarly, pcStream (Mirsky et al. 2015) is also defined to operate in a damped window mode. pcStream dynamically detects and manages temporal contexts by fusing sensor data streams to infer the present concepts and detects new concepts as they emerge.

The sliding window models approach the data stream in a similar way as damped windows but can be seen as stricter. Instead of having a decaying function, the sliding window

is of a fixed size, and all objects within the sliding window have the same level of importance. When an object is added at one end of the window, another object is removed at the other end of the window, providing a window sliding over the stream.

DenStream (Cao et al. 2006) and its extension HDDStream (Ntoutsi et al. 2012), that handles high-dimensional data, follow an online-offline design, and are based on the DBSCAN clustering algorithm. The online procedures of the algorithms produce micro-clusters based on the density, which are later fed to the offline procedures, where the real clusters are created. WCDS (Cardoso et al. 2017) also follows the online-offline approach, creating micro-clusters in the online phase, but the offline phase was based on an agglomerative clustering algorithm to define its top-level clusters.

In contrast, landmark window models divide the data by assigning fixed landmarks where all data between two landmarks is a window. When a landmark is reached and a window ends, the succeeding window starts from that landmark point. A typical approach for landmark window-based clustering algorithms is to use the divided data stream to cluster them separately and use the produced centroids for that segment as representation, usually with partition-based algorithms.

The Stream framework was one of the earliest methods for stream clustering (Guha and Mishra 2016). The data stream is divided into segments, and each segment is clustered by k-median. The produced cluster centers from the segments are added into buckets representing a prototype array, ending up with k_i medians, where i is the number of clustered segments. Whenever the number of stored medians surpass a parameter m, k-medians is run upon the prototype array to produce a median of medians situation. Stream LSearch is an extension of the Stream method, where a more effective subroutine for the underlying k-median was introduced called LSearch (O'callaghan et al. 2002). In 2015, a stream adaptation called StreamKM++ (Anderson and Koh 2015) was proposed to the kmeans++ algorithms. StreamKM++ creates a coreset tree by sampling a subset of the segment and solves the optimization problem on that subset without touching the rest of the segment. These sets are then stored in buffers that are merged whenever a new segment is clustered. StreamXM is a continuation of StreamKM++ and operates similarly, with Xmeans as the underlying clustering algorithm (Anderson and Koh 2015).

DUCstream also divides the stream into segments that are manageable for the system memory, but its underlying structure is instead a density-based algorithm (Gao et al. 2005). DUCstream partitions the data space in units and map the incoming objects in the units; the more mapped objects to a unit, the denser it is. These dense units are then used to perform clustering.

None of the methods mentioned above are explicitly tailored for our specified problem. They aim to model the entire stream with a single clustering solution as best as possible. We instead intend to divide the stream into segments, cluster them separately, and use the clustered segments to see how the stream evolves. With clustering solutions produced of each segment, it is easier to analyze the data between segments and trace how clusters have remained, changed, disappeared, or appeared. We have identified two approaches that similarly address the data stream clustering problem, namely PivotBiCluster (Ailon et al. 2012) and Split-Merge Evolutionary Clustering algorithms (Boeva et al. 2019).

3.3 PivotBiCluster

The first algorithm we compare with is PivotBiCluster (Ailon et al. 2012), an algorithm related to Bipartite Correlation Clustering (BCC) (Amit 2004). BCC builds upon the notion of taking two clustering solutions and combine them into a larger solution. Either by directly combining two clusters from different solutions or dividing a cluster from one solution into several clusters in the other solution. The combination is decided upon the correlation between the clusters of the different clustering solutions.

Referring to our problem statement, located in Sect. 4.1, PivotBiCluster assumes two data segments have been clustered beforehand, e.g., D_0 and D_1, thus producing C_0 and C_1. These two clustering solutions (C_0 and C_1) are then given to PivotBiCluster, which tries to combine them together, creating C'_1, by merging clusters from each solution based on how similar they are to each other. The correlation clustering can be applied over and over; thus, in the formalized problem statement, the PivotBiCluster continues to create a large clustering solution by using C'_1 in combination with C_2 to produce C'_2

One of the drawbacks of the PivotBiCluster algorithm is a lack of the ability to split a cluster into several others in the other clustering solutions. This drawback was the primary motivation of the Split-Merge Evolutionary Clustering algorithm.

3.4 Split-merge evolutionary clustering

The Split-Merge Evolutionary Clustering (Split-Merge Clustering) algorithm builds upon the idea present in BCC clustering algorithms, with the addition of splitting a cluster into multiple clusters in the other clustering solution (Boeva et al.

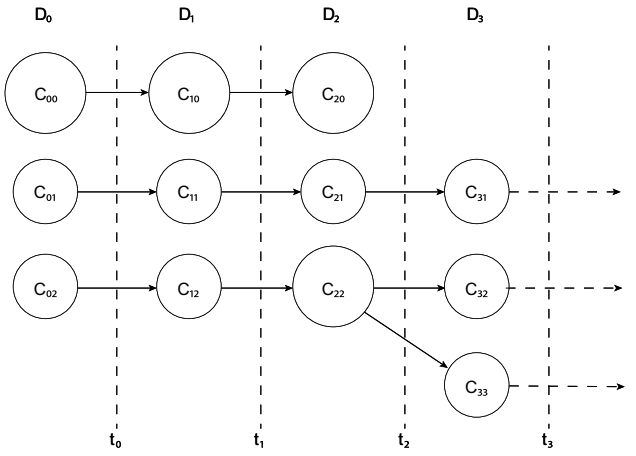

Fig. 1 A schematic illustration of the proposed EvolveCluster. D_i represents data segments at time t_i, C_{ij} represents clusters in clustering solutions C_i of data segments D_i, and t represents individual time segments

2019). This means that if a larger cluster exists within one of the clustering solutions, it can be split up into multiple clusters in the algorithm's output. Similar to the approach of PivotBiCluster, the Split-Merge Clustering algorithm also assumes the incoming data from the data stream D is clustered in advance. Additionally, just as the PivotBiCluster, the clustering can either occur continuously, thus create a final large clustering solution that contains all the data elements from the dataset. Another option can be to take intermediary steps to filter out the data from the previous segment(s) to create a more reflecting model on the present type of data in the data stream.

One of the significant benefits of Split-Merge Clustering, compared to PivotBiCluster, is the splitting of clusters. The authors claim that with that addition, the algorithm is less sensitive to under- and over-clustering of each data segment, as the clusters are now more easily modified over time.

In contrast to our proposed algorithm, which we present in the following section, the Split-Merge Clustering algorithm links the old and new clustering solutions together. Depending on what type of data is being analyzed, this can be counter-productive if the clustering aims not to solely focus on current trends in the data.

4 An evolutionary clustering algorithm

4.1 Problem statement

Let us formalize the evolving data scenario we aim to address. Assume that D is a continuous stream of data, and a vector of features represents each data point. D_0 is the initial data segment which has been partitioned into k clusters, $C_0 = \{C_{00}, \ldots, C_{0k}\}$. Additionally, $D_1, \ldots, D_t$, where $t \to \infty$, are continuous segments of data in the stream to be partitioned. Our objective is to produce a clustering solution, or clustering solutions, modeling how the data evolves.

4.2 EvolveCluster: an evolutionary clustering algorithm

In this section, we formally describe the proposed sequential partitioning algorithm, entitled EvolveCluster. The main idea of EvolveCluster is to allow a continuous data behavior to be easily modeled, by incorporating gained knowledge from the previous data segments, in the form of the cluster centroids, to influence the clustering of the new data segment. Using previous centroids, we can trace how the clusters evolve as the clusters are related over the data segments. Likewise, with each segment being clustered individually, it is easy to identify reoccurring trends between segments and changes in the data. The algorithm idea is schematically illustrated in Fig. 1.

Similar to both PivotBiCluster and Split-Merge Clustering, EvolveCluster divides the data stream into individual data segments. Likewise, the initialization of EvolveCluster requires the first data segment to be clustered in advance. The remaining data segments are, however, clustered within EvolveCluster. Each segment is sequentially clustered, using a partitioning algorithm, with the aid of the cluster centroids (seeds) from the previous segment. EvolveCluster assumes the new data segment contains at least some of the structure from the previous segment by incorporating the clustering structure from the previous segment. The following operations are conducted on each new data segment:

- The data points of the segment is initially clustered by seeding with the cluster centroids of the previous segment;
- The old centroids are removed and any empty clusters are deleted;
- New centroids for the clusters are elected, and the clustering solution is refined.

The refined clustering solution undergoes a "trial-and-error" approach to detect if any clusters should be split into two by applying a 2-means clustering algorithm on a cluster basis. The 2-means clustering algorithm is initialized with the two data points in the cluster that exhibit the furthest distance to each other. If the clustering solution containing the split clusters is deemed the better clustering solution, by a validation measure, it is kept. Otherwise, it is discarded. The algorithmic steps conducted at each data segment are defined in Algorithm 1.

Algorithm 1: EvolveCluster

Input : Data segment D_t, Centroids $\{c_1, \ldots, c_k\} \in C_{t-1}$
Output: Clustering solution C_t

```
C_t ← InitialPartition(D_t, c_1, ..., c_k)
C_t ← RefineSolution(C_t)
S ← SI(C_t)
while SplitPerformed = True do
    SplitPerformed ← False
    for C_ti ∈ C_t do
        C'_t ← Split(C_t, C_ti)
        if S + τ < SI(C'_t) then
            S ← SI(C'_t)
            C_t ← C'_t
            SplitPerformed ← True
            break
        end
    end
end
```

4.3 Computational complexity

In this section, we examine the computational costs of the clustering and splitting operations of the proposed algorithm. Depending on what underlying clustering algorithm is used, the computational complexity will differ. The approach proposed in this study uses k-medoids, a distance based partitioning algorithm. k-medoids requires a distance matrix of size $n \times n$ to be computed, where n is the number of elements. The distance matrix occupies the majority of both the computations and memory consumption of the algorithm, being a complexity of $O(n^2 d)$ and $O(n^2)$, respectively, where d is the feature space dimension.

In this study, we propose the use of k-medoids whose complexity has been thoroughly studied (Schubert and Rousseeuw 2019). We can divide k-medoids into two parts: i) Initialization and ii) Refinement. The initialization according to the original implementation, which opts to identify a beneficial starting point, generates a complexity of $O(n^2 k)$ where k is the number of clusters. When initial medoids are provided, or randomly chosen, the complexity instead becomes $O(nk)$. However, the refinement process remains the same as originally defined, generating a complexity of $O((n-k)^2 ki)$, where i is the number of iterations performed in the refinement. This we can simplify to $O(n^2 ki)$ as $k << n$.

Here, we present the computational complexity of a single iteration of EvolveCluster. Suppose n is the number of data instances in the entire dataset and n' is the number of instances in each data segment, where $n' << n$. The initial clustering occurs in two steps, InitialPartition and RefineSolution, as defined in Algorithm 1 (steps one and two, respectively). InitialPartition assigns each data object in the current segment to the closest centroid, removes the initial centroids, and deletes any empty clusters, with a computational cost of $O(n'k + k + k) \rightarrow O(n'k)$. RefineSolution is a direct implementation of the original k-medoids algorithm, giving a complexity of $O(n'^2 ki)$. The initial clustering of EvolveCluster then becomes $O(n'k + n'^2 ki)$, which can be simplified to $O(n'^2 ki)$.

The split criterion of EvolveCluster is calculated once outside the loop and once for every time a split is performed inside the loop. In the proposed approach, we use the SI as our measure for the split criterion, which has a computational complexity of $O(2n' + n'^2)$. EvolveCluster splits a cluster by first identifying the two elements that are the furthest apart, $O(n'^2)$. k-medoids is then used with $k = 2$ with the two identified elements as initial centroids, i.e. $O(n'k + n'^2 ki)$. A single iteration of the splitting loop then becomes $O(n'^2 + n'k + n'^2 ki + 2n' + n'^2)$, which we can simplify to $O(n'^2 ki)$.

As each cluster in the produced clustering solution C_t is split at least once, the lower bound of iterations for the splitting part of EvolveCluster becomes k times. This gives the lower bound for splitting to be $O(k(n'^2 ki)) \rightarrow O(n'^2 k^2 i)$. The upper bound, on the other hand, is dramatically higher. In the worst case, a split is performed in every iteration which causes the final clustering solution to consist solely by singleton clusters, i.e., n' iterations. The upper bound then becomes $O(n'(n'^2 ki)) \rightarrow O(n'^3 ki)$. Finally, the total complexity for each increment, with the inclusion of the distance matrix calculation, of the EvolveCluster algorithm is $O(n'^2 ki + n'^2 k^2 i + n'^2 d) \rightarrow O(n'^2(k^2 i + ki + d)) \rightarrow O(n'^2(k^2 i + d))$. If we include the upper bound calculation, the complexity of EvolveCluster becomes $O(n'^2 ki + n'^3 ki + n'^2 d) \rightarrow O(n'^3 ki)$.

Table 1 Information regarding number of features and instances of each dataset in their original form. The number of instances in the DELMH dataset are individual measurements from 71 up to 2940 concurrent households over 21 years, varying between 1 measurement up to 12076 per household

Dataset	No. features	No. instances
S1	2	5000
Covertype	54	581012
DELMH	1	3341726

Similarly to EvolveCluster, the Evolutionary Split-Merge Clustering bases its complexity on the underlying clustering algorithm (Boeva et al. 2019). Evolutionary Split-Merge Clustering adds the additional computational overhead $O((k' + k')n')$. In combination with k-medoids, its complexitiy becomes $O((k' + k')n' + n'^2ki) \rightarrow O(n'^2ki)$, which is in line with the produced lower bound complexity of EvolveCluster. The auhors of PivotBiCluster, on the contrary, have not proposed their complexity calculations in the clustering scenario (Ailon et al. 2012). Thus, we have no direct comparisons to perform.

5 Data and experimental designs

We perform two sets of experiments to investigate the effectiveness of EvolveCluster. The first experiment compares EvolveCluster and two similar clustering algorithms on three different datasets to analyze their differences and performances. In our second experiment, we analyze how EvolveCluster handles different concept drift scenarios by generating a synthetic data stream.

5.1 Experiment 1: comparative analysis

5.1.1 Data

We evaluate and compare the performance of the proposed EvolveCluster algorithn to two other clustering algorithms (PivotBiCluster and Split-Merge Clustering) on three different datasets, explained in Table 1. The first is the S1 dataset, a 2-dimensional synthetic dataset created by the authors of (Fränti and Virmajoki 2006). This dataset is chosen to investigate the algorithms ability to identify new clusters as they arrive in the data stream, and how they manage with regard to clustering a constant type of behavior over time.

The second dataset is a subset of the Covertype dataset, available at the UCI repository (Hettich and Bay 1999). The motivation behind the use of this dataset is mainly to have a direct comparison to both the PivotBiCluster and Split-Merge Clustering algorithms, as the authors of the latter algorithm have performed experiments upon it in their paper (Boeva et al. 2019). However, it is also chosen due to its larger number of data points in combination with a higher dimensionality of its features compared to the S1 dataset.

Finally, the third dataset is a real world electricity consumption dataset, the Domestic Electrical Load Metering, Hourly Data (DELMH) (Toussaint 2019). DELMH contains consumption from a large number of households and metering stations in South Africa covering the period from 1994 to 2014, with measurements taken up to every 5 minutes. It is worth noting that the single household with the most prolonged consumption period amounts to roughly two years worth of consumption. This type of dataset is one of the main target areas for our proposed algorithm.

All information about the used datasets in their original form is presented in Table 1.

5.1.2 Data pre-processing

5.1.2.1 S1 dataset The S1 dataset is divided into five equal parts, each part consisting of 1'000 elements. We do, however, create two distinct experimental datasets from the S1 dataset. The first dataset keeps the original format where each cluster appears one by one in order, but the other dataset is modified such that all data segments contain the same ratio of all clusters (see below). The two features are normalized in the $[0 - 1]$ range by a Min-Max feature scaling. Each feature value is subtracted by the minimum value of that feature (X_{min}), and then divided by the difference between the minimum and maximum value (X_{max}) of the feature, i.e.,

$$x' = \frac{x - X_{min}}{X_{max} - X_{min}}.$$

with the S1 dataset, we want to allow the algorithms to showcase how they handle two aspects of data stream clustering. The first is to discover new clusters as they appear in the data. By keeping the S1 dataset in its original state, each cluster appears in the data stream one after another. Between each segment in the original dataset, 2 to 3 clusters disappear, and 2 to 3 new clusters appear.

The second aspect is to model a continuous set of behaviors in the data stream over time. We simulate this aspect by dividing the data points in each cluster evenly between each segment. Thus, 20% of each clusters' data points are located in D_0. D_1 consists of the next 20% appear and so on. From here on and forward, we denote this dataset as the *continuous S1* dataset.

5.1.2.2 Covertype dataset To mimic the experiments of the authors of the Split-Merge Evolutionary Clustering algorithm, we perform the same steps of pre-processing and segmentation of the Covertype Dataset (Boeva et al. 2019). A subset of 50'000 elements is randomly chosen out of the

Table 2 Information regarding number of features and data points in each data segment of all datasets after pre-processing

Dataset	No. features	No. instances				
		D_0	D_1	D_2	D_3	D_4
S1 original	2	1000	1000	1000	1000	1000
S1 continuous	2	995	1000	1000	1000	1005
Covertype	14	35000	15000	–	–	–
DELMH	24	198	74	75	74	75

581'012 and 14 out of the 54 features is chosen, excluding all binary features regarding the soil type. Each feature is standardized using the z-score, where each feature is subtracted by their mean value ($\bar{x}$) and divided by the standard deviation (σ), i.e.

$$z = \frac{x - \bar{x}}{\sigma}.$$

The 50'000 data points are divided into 2 segments in a 70-30 split, creating D_0 with 35'000 elements and D_1 with 15'000 elements.

5.1.2.3 DELMH dataset For the DELMH dataset, we first divide all available measurements into their corresponding households. All measurements are combined into daily profiles, such that each daily profile contains measurements from 00:00 to 23:59. Every daily profile consists of 24 data points, where each data point in the profile represents the aggregated consumption of each hour. If any profile contains a measure that is indicated to be invalid, the entire daily profile is dismissed for further use. All profiles undergo the same z-score standardization as mentioned above for the Covertype dataset. However, instead of applying it on a feature level, we apply it to each individual profile. Each standardized profile represents the shape of the electricity consumption and disregards the actual amplitude of the electricity consumption.

We identify the 10 households with the largest number of daily profiles, and choose one of them to represent the use case for our algorithm. The chosen household consists of 496 daily profiles after the pre-processing stage, starting from 1997-12-31 and ending on 1999-05-06. The final 496 profiles are divided into 5 segments, where the first segment (D_0) contains 198 elements, corresponding to 40% of the number of profiles. The remaining 4 segments contain 74-75 profiles each, representing 15% of the total number of profiles.

A summary of all dataset information after the pre-processing and modification is located in Table 2.

5.2 Experiment 2: concept drift analysis

In the second experiment, we specifically investigate how EvolveCluster performs in an evolving data stream scenario. This experiment is conducted on generated synthetic data to make sure a ground truth is available, the data contains concept drift, and when the concept drifts occur. We created a Radial Basis Function Generator (RBFGenerator) based on the implementations available at MOA (Bifet et al. 2010a) and scikit-multiflow (Montiel et al. 2018). Both MOA and scikit-multiflow implementations provide evolving data streams that contain a constant drift of clusters, where each cluster centroid moves as time progress. However, scikit-multiflow's implementation does not contain any creation or deletion of clusters. Conversely, MOA includes options of specific events, such as cluster creation and deletion, but cannot export its stream if more than one additional cluster is created in the stream. Thus, we have created an RBFGenerator that produces evolving data streams with no limitation on the functionalities mentioned.

The produced data streams consist of 10'000 data points with 2 features. Each stream is initialized with the same random seed but has different seeds for generating data points and cluster events. When 2'500 data points have been produced, an event occurs in the stream. Additional events then occur after each 2'000th data point, i.e. at 4'500, 6'500, and 8'500. An event is randomly chosen out of two options, creation or deletion. The streams begin with 5 clusters and are allowed to vary between 2 and 8. The cluster centers are limited to the [0-1] domain in both features. The cluster centers' speed is randomly chosen but limited to 0.0001 per instance created in the stream. Similarly, the radius of each cluster is randomly chosen but limited to be 0.02 ± 0.005.

The data streams are divided into five segments each, creating data segments D_0, D_1, D_2, D_3, D_4. Each segment from D_1 and onwards contains an event. To initialize EvolveCluster on the produced data stream, we used the cluster labels given by the RBFGenerator on the D_0 segments and then calculated the centermost point (i.e., medoid) in each cluster to use as cluster centers.

5.3 Evaluation and validation

In this study, we combine both internal and external cluster validation measures to assess the results from both experiments. In the first experiment, both the Covertype and the S1 datasets have ground truth labels, allowing us to use external validation measures. For these two datasets, we have used

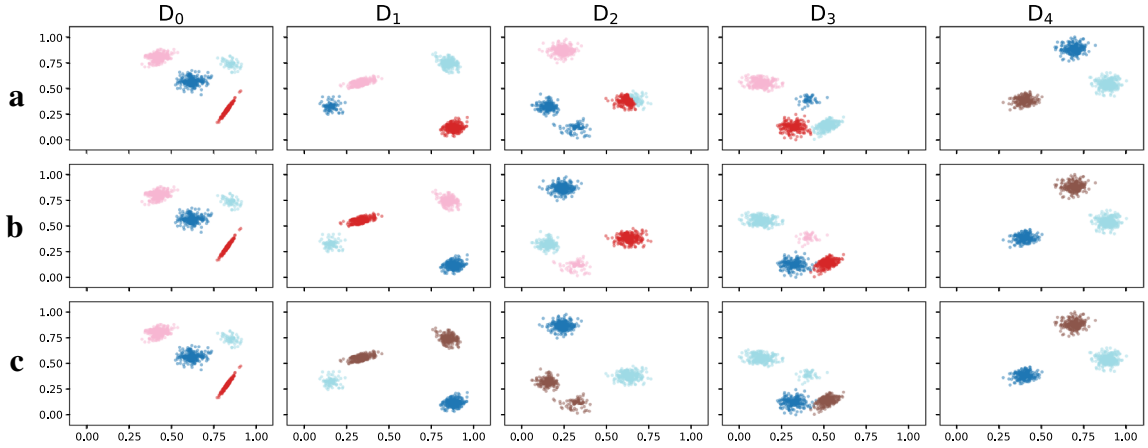

Fig. 2 The clustering solutions obtained on each data segment D_0, D_1, D_2, D_3, D_4 of the original S1 dataset. The results generated by the three studied algorithms are depicted as follows: **a** EvolveCluster, **b** Split-Merge Clustering, and **c** PivotBiCluster. Each color of the data points in the figures represents a single cluster within that segment

Table 3 Results from the validation measures F_1, JI, and SI on the original S1 dataset, where previous data segments are discarded before evaluating, for all three clustering algorithms

		D_0	D_1	D_2	D_3	D_4
EvolveCluster	F_1	1	1	0.770	0.997	1
	JI	1	1	0.681	0.993	1
	SI	0.826	0.879	0.649	0.750	0.867
Split-Merge	F_1	1	1	1	1	1
	JI	1	1	1	1	1
	SI	0.826	0.879	0.848	0.747	0.867
PivotBiCluster	F_1	1	0.908	0.947	0.978	1
	JI	1	0.856	0.909	0.958	1
	SI	0.826	0.538	0.824	0.700	0.867

both the F_1 and JI measures to evaluate how the three studied clustering algorithms perform regarding the known structure of the datasets. Additionally, we apply the SI to assess the compactness and separation of the produced clusters. The same applies to the data stream we generate for the second experiment; thus, we apply the same validation measures as for Covertype and S1. The DELMH dataset in the first experiment, on the other hand, has no ground truth available, causing us to focus solely on the internal cluster evaluation measures. We instead apply the SI and IC-av measures to evaluate how good the produced clustering solutions are.

To show how the three algorithms in the first experiment perform over time, we calculate the evaluation metrics for each individual data segment of the datasets in three ways:

1. Only the data arriving in the current segment is used for calculating the scores, disregarding how the old data segment has been altered and merged.
2. Each segment is calculated in combination with the previous data segment.
3. For some of the experiments, we also evaluate all the data segments up until that point in time, e.g., when we reach D_3, we include D_0, D_1 and D_2 in the evaluation.

5.4 Implementation and availability

PivotBiCluster and Split-Merge clustering operate by taking existing clustering solutions and combine them into a larger clustering solution. Both algorithms require each data segment to be clustered beforehand and combine the produced clustering solutions to create a combined version. EvolveCluster, on the other hand, focuses solely on clustering the current data segment and only incorporates the cluster centroids of the previous data segment to produce the next, disregarding the old clustering solution after its initial clustering. To compare the results of these three algorithms, we have implemented two additional variations of the PivotBiCluster and Split-Merge Clustering algorithms. The first variation is implemented so that after each segment is clustered, data belonging to the previous data segment is removed, similar to the EvolveCluster algorithm. The second

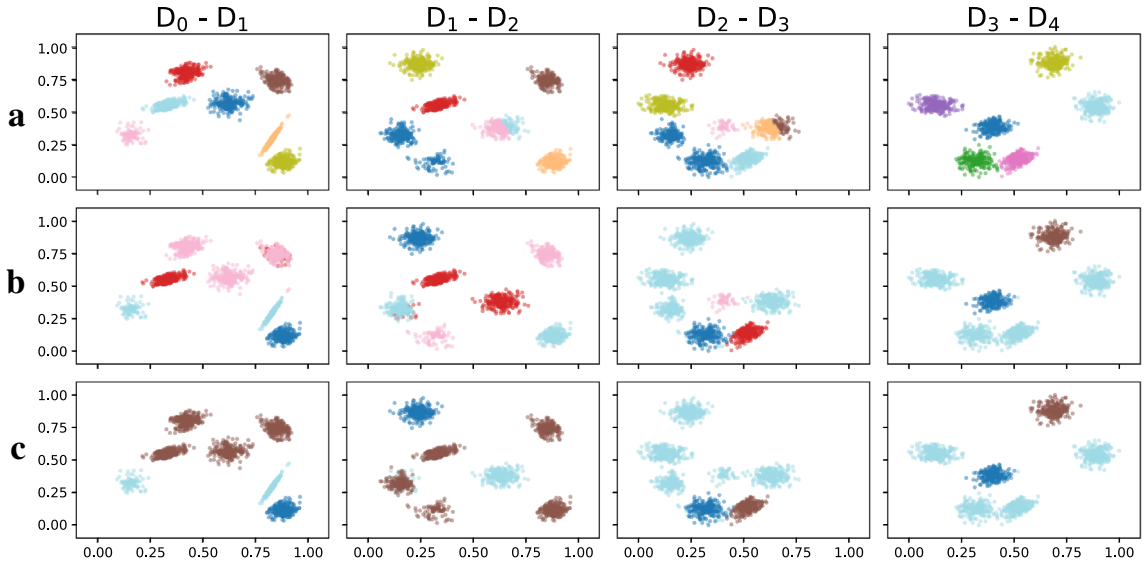

Fig. 3 The clustering solutions obtained on each data segment D_1, D_2, D_3, D_4 of the original S1 dataset, where the previous segment is included. The results generated by the three studied algorithms are depicted as follows: **a** EvolveCluster, **b** Split-Merge Clustering, and **c** PivotBiCluster. Each color of the data points in the figures represents a single cluster within that segment

Table 4 Results from the validation measures F_1, JI, and SI on the original S1 dataset, where the previous data segment is kept during evaluation, for all three clustering algorithms

		D_0–D_1	D_1–D_2	D_2–D_3	D_3–D_4
EvolveCluster	F_1	1	0.875	0.854	0.998
	JI	1	0.827	0.797	0.995
	SI	0.783	0.716	0.567	0.793
Split-Merge	F_1	0.824	0.797	0.816	0.778
	JI	0.735	0.687	0.750	0.705
	SI	0.313	0.365	0.129	0.103
PivotBiCluster	F_1	0.758	0.764	0.750	0.778
	JI	0.672	0.679	0.664	0.705
	SI	0.414	0.176	0.222	0.103

variation retains the previous data segment as the clustering progresses but is removed before the next data segment is clustered.

For the first experiment, all three clustering algorithms are initialized using the same clustering solution C_0. Providing all algorithms with the same starting point gives us insight into how they produce clustering solutions for each data segment. For PivotBiCluster and Split-Merge Clustering, we have to cluster $D_1, D_2, \ldots, D_n$ before using them in the algorithms. In the S1 and Covertype experiments, we use the ground truth labels in conjunction with the NearestCentroid classifier (Tibshirani et al. 2002) to find the centroids to be able to cluster C_2' and onwards. For DELMH, we employ k-medoids to do the initial clustering of $C_1, C_2, \ldots, C_n$. This initial clustering is run for 1'000 iterations for each segment and the number of clusters between 2 and 10. Each clustering solution is evaluated via SI and IC-av. Empirically we choose upon the produced clustering solutions as the input for the PivotBiCluster and Split-Merge Clustering algorithms. Finally, all the experiments use the Euclidean Distance as the dissimilarity measure.

Table 5 Results from the validation measures F_1, JI, and SI on the continuous S1 dataset, where the all previous data segments are kept during evaluation, for the PivotBiCluster and Split-Merge Clustering algorithm

		D_0	D_0–D_1	D_0–D_2	D_0–D_3	D_0–D_4
Split-Merge	F_1	1	0.824	0.660	0.760	0.691
	JI	1	0.735	0.534	0.712	0.646
	SI	0.826	0.313	0.121	−0.102	−0.166
PivotBiCluster	F_1	1	0.758	0.613	0.760	0.691
	JI	1	0.672	0.564	0.712	0.646
	SI	0.826	0.414	0.267	−0.102	−0.166

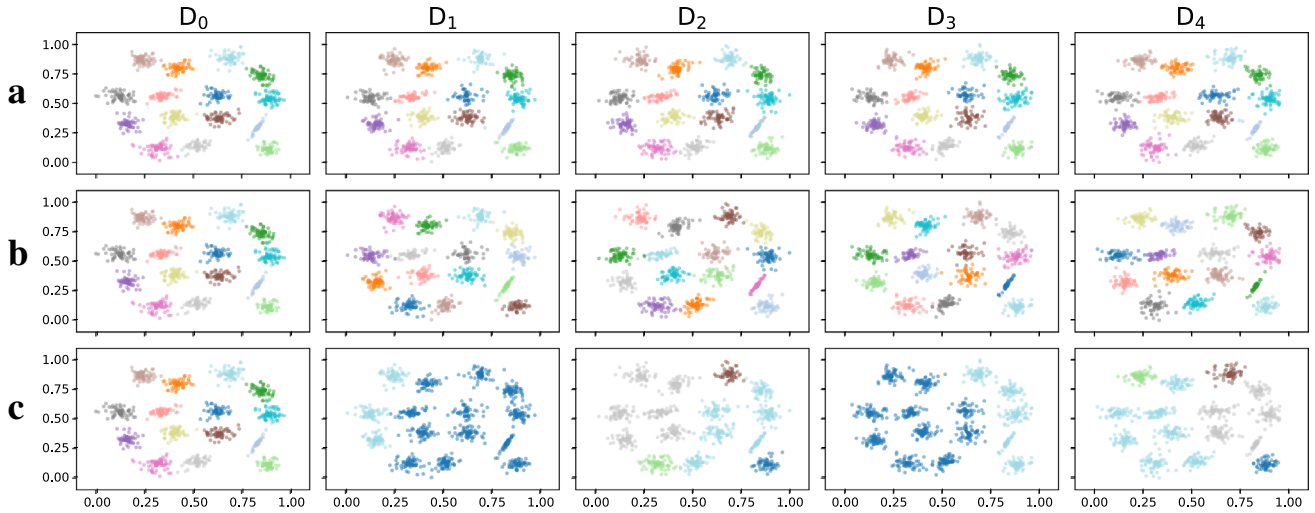

Fig. 4 The clustering solutions obtained on each data segment D_0, D_1, D_2, D_3, D_4 of the continuous S1 dataset. The results generated by the three studied algorithms are depicted as follows: **a** EvolveCluster, **b** Split-Merge Clustering, and **c** PivotBiCluster. Each color of the data points in the figures represents a single cluster within that segment

Table 6 Results from the validation measures F_1, JI, and SI on the continuous S1 dataset, where previous data segments are discarded before evaluating, for all three clustering algorithms

		D_0	D_1	D_2	D_3	D_4
EvolveCluster	F_1	0.997	0.990	0.993	0.996	0.992
	JI	0.994	0.980	0.986	0.992	0.984
	SI	0.722	0.715	0.698	0.716	0.708
Split-Merge	F_1	1	1	1	1	1
	JI	1	1	1	1	1
	SI	0.719	0.710	0.693	0.713	0.705
PivotBiCluster	F_1	1	0.334	0.720	0.269	0.722
	JI	1	0.213	0.670	0.158	0.672
	SI	0.719	0.271	0.208	0.354	0.270

All experiments and algorithms are implemented in Python 3.6.10 and are available for download here[1].

6 Results and analysis

In this section the results from all experiments are presented and discussed following the order in which the datasets have been explained in Sect. 5.1.1.

6.1 Original S1 dataset

6.1.1 Original S1

The results from the experiment on the original S1 dataset are presented in Fig. 2 and Table 3. In Fig. 2, we observe that the EvolveCluster and Split-Merge Clustering algorithms are more proficient than PivotBiCluster in identifying new clusters when they arrive in the data stream. This is further strengthened in Table 3, where overall scores suggest that PivotBiCluster performs slightly worse compared to the other two algorithms. However, as it can be seen in Fig. 2a (under the D_2 header) we observe that EvolveCluster shows a difficulty in merging clusters together when they are initiated closely together. This result is logical, since EvolveCluster has no specific merge criterion or dedicated process for merging more than if the initial clustering of each segments produce empty clusters they are removed. It is also interesting to notice that Split-Merge Clustering fully follows the true clustering of the data points, up to the point that even data points that are overlapping into another cluster is correctly classified.

To further investigate how the three algorithms operates, we include the previous data segments for each clustering solution as explained in Sect. 5.3. These results are presented

[1] https://github.com/christiannordahl/EvolveCluster

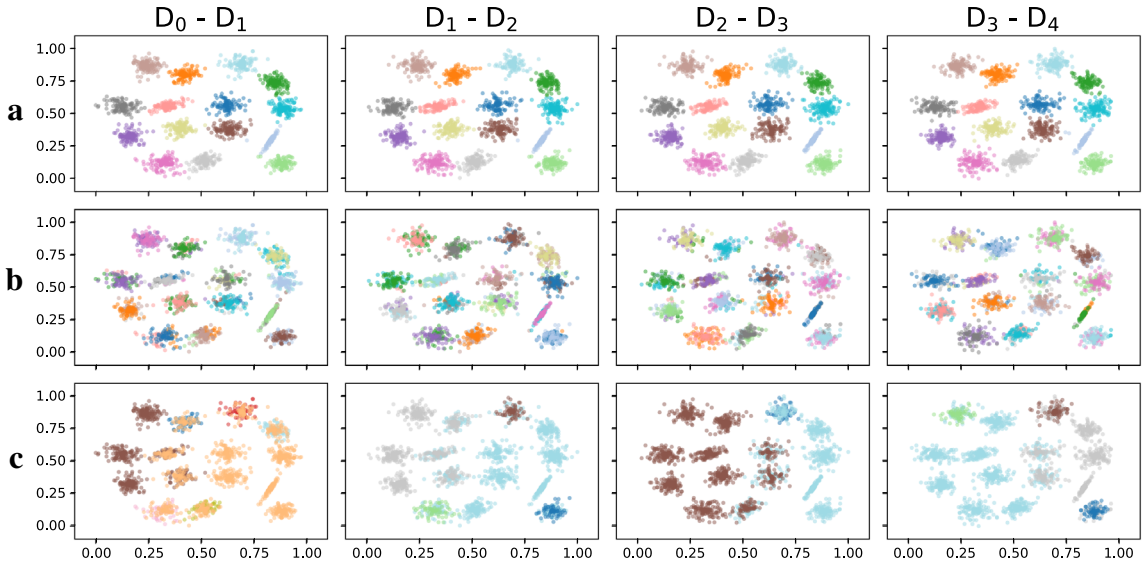

Fig. 5 The clustering solutions obtained on each data segment D_1, D_2, D_3, D_4 of the continuous S1 dataset, where the previous segment is included. The results generated by the three studied algorithms are depicted as follows: **a** EvolveCluster, **b** Split-Merge Clustering, and **c** PivotBiCluster. Each color of the data points in the figures represents a single cluster within that segment

Table 7 Results from the validation measures F_1, JI, and SI on the continuous S1 dataset, where the previous data segment is kept during evaluation, for all three clustering algorithms

		D_0–D_1	D_1–D_2	D_2–D_3	D_3–D_4
EvolveCluster	F_1	0.993	0.991	0.995	0.994
	JI	0.987	0.983	0.989	0.988
	SI	0.718	0.707	0.707	0.711
Split-Merge	F_1	0.511	0.518	0.514	0.52057
	JI	0.346	0.354	0.351	0.358
	SI	−0.113	−0.141	−0.121	−0.153
PivotBiCluster	F_1	0.564	0.517	0.396	0.514
	JI	0.410	0.370	0.267	0.368
	SI	−0.109	−0.065	0.177	0.025

in Fig. 3 and Table 4. It is clear that most of the structure from the previous data segment is lost when combining clustering solutions, both by the PivotBiCluster and, especially, the Split-Merge Clustering algorithm. Split-Merge Clustering has a perfect score in both F_1 and JI when each segment is validated separately, which is to be expected. Each of the segments are clustered in advance and are following the ground truth labels from the dataset. Comparing the scores from the evaluation measures in Table 4, we also observe a significant decrease of all scores for all three algorithms. The drop for EvolveCluster, however, is minor in comparison.

Furthermore, in Table 5 we show the results for PivotBiCluster and Split-Merge Clustering when all previous segments are retained in the clustering solutions. It is clear that as the algorithms progress through the data segments, more clusters from the previous segments get merged into one large cluster for both PivotBiCluster and Split-Merge Clustering.

6.1.2 Continuous S1 dataset

In this subsection, we present the results obtained from the continuous version of the S1 dataset, presented in Fig. 4 and Table 6. We observe that the performance of PivotBiCluster in this scenario dramatically decreases. In the first iteration, when D_1 is clustered, there is an instant decrease in the number of clusters (see Fig. 4). The clustering solution should

Table 8 Results from the validation measures F_1, JI, and SI on the continuous S1 dataset, where the all previous data segments are kept during evaluation, for the PivotBiCluster and Split-Merge Clustering algorithm

		D_0	D_0–D_1	D_0–D_2	D_0–D_3	D_0–D_4
Split-Merge	F_1	1	0.511	0.398	0.340	0.343
	JI	1	0.346	0.251	0.207	0.210
	SI	0.719	−0.113	−0.228	−0.206	−0.146
PivotBiCluster	F_1	1	0.564	0.346	0.250	0.303
	JI	1	0.410	0.214	0.143	0.180
	SI	0.719	−0.109	−0.067	0.227	−0.166

Table 9 Results from the validation measures F_1, JI and SI on the Covertype dataset for all three algorithms, on both the individual (D_0 and D_1) and combined ($D_0 - D_1$) data segments

		D_0	D_1	D_0–D_1
EvolveCluster	F_1	1	0.422	0.539
	JI	1	0.275	0.436
	SI	0.063	−0.007	−0.040
Split-Merge	F_1	1	1	0.754
	JI	1	1	0.656
	SI	0.063	0.062	0.034
PivotBiCluster	F_1	1	0.905	0.903
	JI	1	0.849	0.848
	SI	0.063	0.192	0.194

Table 10 Number of clusters for each algorithm on the Covertype dataset, on both the individual (D_0 and D_1) and combined ($D_0 - D_1$) data segments

Algorithm	D_0	D_1	$D_0 - D_1$
EvolveCluster	7	3	7
Split-Merge	7	7	7
PivotBiCluster	7	5	5
Ground truth	7	7	7

contain 15 clusters, as is the case for both EvolveCluster and Split-Merge Clustering. PivotBiCluster instead opts to reduce the number of clusters to 2, showing a clear case of under-clustering. In contrast, both EvolveCluster and Split-Merge Clustering show that they can cluster a continuous set of behaviors over time.

Similarly to the original S1 dataset, when the data segments are merged, it is apparent that Split-Merge Clustering fully adapts the previous data segment's clustering solutions to the new segments. Figure 5b shows that many of the data points are incorrectly clustered for Split-Merge Clustering. PivotBiCluster has a similar experience when the segments are merged, as on the separated segments, a clear result of under-clustering. EvolveCluster, on the other hand, manages to retain a higher performance with validation measure scores on par with the non-merged segments. These results are further emphasized by looking at Table 7.

Table 12 Results from the validation measures Silhouette Index and Average Intra-Cluster Distance on the DELMH dataset, where the previous data segment is kept during evaluationg, for all algorithms

		D_0–D_1	D_1–D_2	D_2–D_3	D_3–D_4
EvolveCluster	SI	0.071	0.057	0.067	0.035
	IC-av	1716	1268	1107	1147
Split-Merge	SI	0.019	−0.012	0.019	0.065
	IC-av	1727	1256	1079	1054
PivotBiCluster	SI	0.002	−0.015	0.040	0.074
	IC-av	1778	1245	1080	1163

Only EvolveCluster manages to produce similar results as when the segments are not merged. Both PivotBiCluster and Split-Merge Clustering show a drastic decrease in all three measures, with SI even producing negative numbers. Finally, when we include all the previous data segments in both the clustering and evaluation, as shown in Table 8, the produced clustering solutions of PivotBiCluster and Split-Merge Clustering are both continuing the declining trend.

Figures 10 and 11, available in the Appendix, show the results produced on both the original and the continuous S1 datasets when all segments are combined.

6.2 Covertype dataset

The results produced by the three clustering algorithms on the Covertype dataset are shown in Table 9. As can be seen, PivotBiCluster outperforms both the Split-Merge Clustering and EvolveCluster algorithms in the case of the $D_0 - D_1$ setup in the table. These results are in line with the results presented in (Boeva et al. 2019). It is interesting to notice the difference between EvolveCluster and Split-Merge Clustering on what scores their clustering solutions obtain in the D_1 column compared to the $D_0 - D_1$ column. Evidently, EvolveCluster does not manage to cluster the D_1 dataset as proficiently as the Split-Merge Clustering algorithm and the final score when the data segments are merged is helped by the clustering from D_0. Additionally, as the results generated on the S1 datasets show (see Figs. 3 and 5), when the two data segments are combined, it is clear that the clustering solution on D_0 given to the

Table 11 Results from the validation measures Silhouette Index and Average Intra-Cluster Distance on the DELMH dataset, where the all previous data segments are discarded before evaluating, for all algorithms

		D_0	D_1	D_2	D_3	D_4
EvolveCluster	SI	0.080	0.119	0.115	0.054	0.066
	IC-av	1189	672	602	570	619
Split-Merge	SI	0.080	0.105	0.150	0.077	0.116
	IC-av	1189	673	592	588	647
PivotBiCluster	SI	0.080	0.119	0.133	0.076	0.116
	IC-av	1189	661	591	595	647

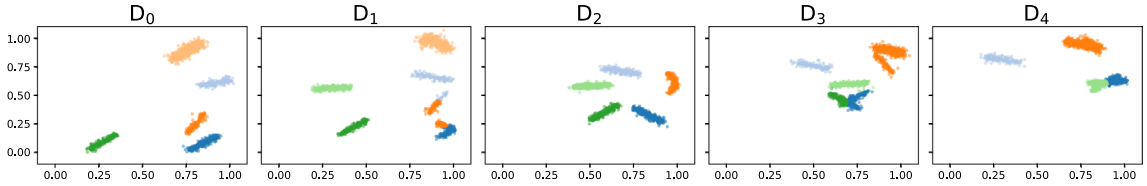

Fig. 6 The clustering solutions obtained on each data segment D_0, D_1, D_2, D_3, D_4 on the data stream generated by our RBFGenerator. Each color of the data points in the figures represents a single cluster within that segment

Table 13 Results from the validation measures F_1, JI and SI on the data stream generated by our RBFGenerator

	D_0	D_1	D_2	D_3	D_4
F_1	1	0.854	1	0.842	0.764
Jaccard	1	0.796	1	0.760	0.662
Silhouette	0.745	0.712	0.697	0.600	0.695

Table 14 Results from the validation measures F_1, JI and SI on the data stream generated by our RBFGenerator

	D_0	D_1	D_2	D_3	D_4
F_1	1	0.903	0.822	0.650	0.662
Jaccard	1	0.857	0.733	0.514	0.566
Silhouette	0.745	0.685	0.576	0.510	0.572

Split-Merge Clustering algorithm is crippled when Split-Merge Clustering clusters the D_1 segment.

Furthermore, in Table 10, we present the number of clusters in each data segment produced by the three algorithms. We can see that EvolveCluster produces far too few clusters when clustering D_1, with only three out of the existing seven. Similarly, PivotBiCluster is under-clustering D_1 with five out of the seven clusters, partly why higher validation measures are obtained for its solution. It is only Split-Merge Clustering that manages to retain all seven clusters. These results follow the previous results on the S1 datasets, where Split-Merge Clustering consistently adapts the clustering solutions from previous segments to the new.

It is also interesting to see that all the produced clustering solutions on the Covertype dataset produce low SI scores. The two aspects that SI concerns are the compactness of the produced clusters and the separation between them. Covertype contains many features and has clusters that overlap each other in some of the features, causing SI to produce lower scores for the clustering solutions. This is evident when we compare the scores of F_1, JI, and SI for all three algorithms, but especially for both Split-Merge Clustering and PivotBiCluster. PivotBiCluster manages to produce an F_1 score of 0.903 and a JI score of 0.848 while only having a SI score of 0.194 (Table 9).

6.3 DELMH dataset

The results for the DELMH dataset are presented in Tables 11 and 12. It is apparent in both tables that all three algorithms produce clustering solutions with much lower SI scores compared to the previous experiments. This is partly because of the difficulty in clustering this dataset. Most of the daily profiles in the dataset are similar to each other. During the majority of the day, there is no actual consumption of electricity. When the residents are out of their homes, only minor consumptions, such as household appliances' idle consumption, are drawn. Similarly, when the residents are asleep, only the idle consumptions are drawn.

In Table 12, we can see that for all segments up to $D_2 - D_3$, the EvolveCluster algorithm performs better in terms of the SI, but IC-av suggests there is no such clear distinction. In the final data segment, it is interesting to see that both PivotBiCluster and Split-Merge Clustering produce higher SI scores than EvolveCluster, and for Split-Merge Clustering, there is also a significantly better IC-av score. However, similarly to the results of the Covertype experiments, both SI and IC-av indicate that the produced clustering solutions are pretty poor. Based on the nature of the data, the electricity consumption of an individual household, it is natural that the produced clustering solutions are deemed

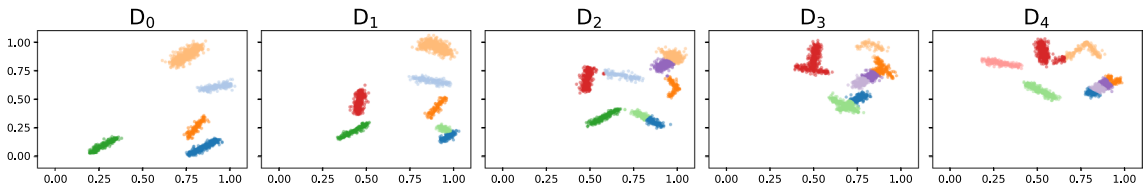

Fig. 7 The clustering solutions obtained on each data segment D_0, D_1, D_2, D_3, D_4 on the data stream generated by our RBFGenerator. Each color of the data points in the figures represents a single cluster within that segment

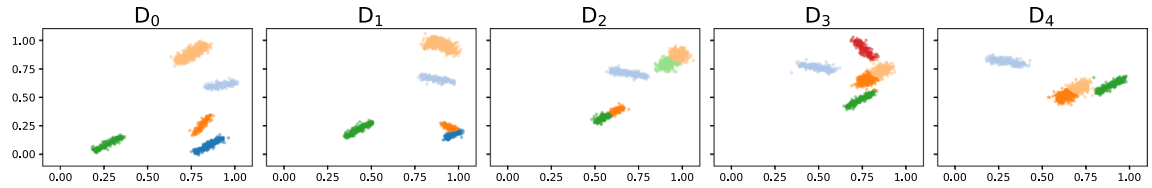

Fig. 8 The clustering solutions obtained on each data segment D_0, D_1, D_2, D_3, D_4 on the data stream generated by our RBFGenerator. Each color of the data points in the figures represents a single cluster within that segment

Table 15 Results from the validation measures F_1, JI and SI on the data stream generated by our RBFGenerator

	D_0	D_1	D_2	D_3	D_4
F_1	1	0.865	0.733	0.863	0.831
Jaccard	1	0.800	0.600	0.794	0.746
Silhouette	0.750	0.718	0.572	0.589	0.591

Table 16 Results from the validation measures F_1, JI and SI on the data stream generated by our RBFGenerator

	D_0	D_1	D_2	D_3	D_4
F_1	1	0.861	0.802	0.755	1
Jaccard	1	0.799	0.703	0.643	1
Silhouette	0.749	0.640	0.664	0.657	0.802

poorly. Many of the daily profiles contain similar values and shapes, making it hard to distinguish between them.

6.4 Concept drift analysis

In this subsection, we present the results of the second experiment, where we investigate how EvolveCluster manages to handle concept drift. In Figs. 6, 7, 8, and 9, we have 4 data streams that are initialized identically but have different continuations. The corresponding validation measures are presented in Tables 13, 14, 15, and 16, respectively.

In the figures, we see constant incremental concept drift in each of the data streams. All clusters move slightly at every new data point, creating oval rather than spherical shaped clusters. We can see that EvolveCluster can model these incremental changes of each cluster, especially when the clusters have some distance between each other. However, when a cluster centroid reaches the boundary of the feature space, its direction is immediately changed to keep the cluster within the boundaries. For instance, in the bottom right corner of Fig. 8 in segment D_1, we identify that the blue cluster has reached the boundary and changed its direction. EvolveCluster assigns the data points belonging to the cluster after the directional change to a new cluster as it is no longer spherical.

In each stream, we can also observe the creation or deletion of clusters in each of the segments from D_1 and onwards. When new clusters appear in the stream, EvolveCluster tends to manage the addition by applying a split. However, when the clusters overlap, such as the light blue cluster in segment D_2 of Fig. 9, the new cluster is not immediately identified.

EvolveCluster relies on the transition between segments to manage the merging of clusters, so we can notice cases of over-clustering in some segments. For instance, in segments D_3 and D_4 of Fig. 7, a single cluster is divided into 3-4 separate clusters on the right side of the figures. When many clusters are close to each other, and then some of them disappear, we can observe EvolveCluster struggling to merge them. When it does not manage to merge clusters, it cascades further to not splitting the neighboring clusters if necessary accurately. This is evident in the upper right corner of the same figure, where the pale orange and red clusters likely should be divided into three.

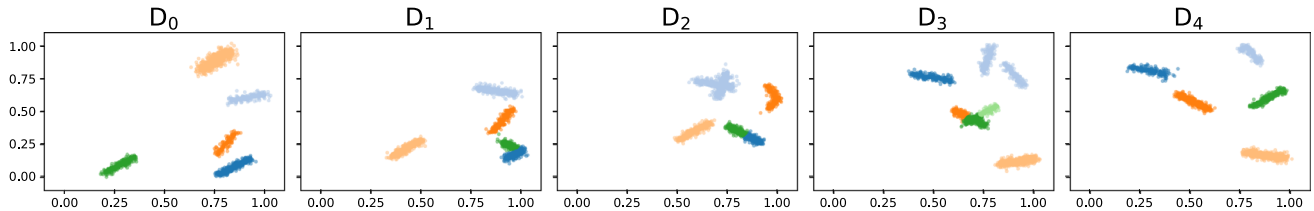

Fig. 9 The clustering solutions obtained on each data segment D_0, D_1, D_2, D_3, D_4 on the data stream generated by our RBFGenerator. Each color of the data points in the figures represents a single cluster within that segment

7 Discussion

7.1 EvolveCluster properties

One of the significant benefits of the proposed EvolveCluster algorithm is its simplicity in using prior knowledge about the previous data segments. With previous centroids, EvolveCluster influences the clustering of the new segment to follow the same structure as the previous. The two aspects we have investigated using the S1 dataset seem to be handled proficiently by EvolveCluster. When we cluster the original S1 dataset, as shown in Fig. 3, EvolveCluster discovers new clusters while maintaining the old clusters still present in the data. However, in Fig. 3a under D_2, we also identify the limitations of the proposed algorithm. When two clusters are close and should be merged, it may result in an over-clustered solution. As there is no specific merge criterion for the algorithm, this is not a surprising result.

Furthermore, as shown by the results from the clustering of the continuous S1 dataset, we can see that it is easy for EvolveCluster to cluster similar behaviors that repeatedly occur in multiple segments. Contrary to both PivotBiCluster and Split-Merge Clustering, it is easy to map and identify trends that occur between segments. When PivotBiCluster and Split-Merge Clustering combine the old data segment with the new, it is clear that the relationships of the clusters between the segments are lost; thus, providing difficulties in analyzing the multiple segments and their trends.

7.2 Comparison to other evolving clustering algorithms

PivotBiCluster is continuously under-clustering the S1 datasets in this study. This can specifically be seen in Fig. 4 and its corresponding Table 6. We can observe here that in the first iteration of the clustering process, PivotBiCluster reduces the number of clusters from 15 to 2 and continues to under cluster for the remainder of that experiment. This is partly why PivotBiCluster achieves a higher score on both the F_1 and Jaccard measures.

With the variation of discarding the previous data segment in its produced clustering solutions, the Split-Merge Clustering algorithm seems to perform better than EvolveCluster in all the experiments in this study. Split-Merge Clustering never performs any modifications to the new clustering solution but instead combines the previous and the new solutions by morphing the old to align with the new. Since all the clustering solutions provided to Split-Merge Clustering are either based on the ground truth labels (S1 and Covertype) or extensively clustered beforehand (DELMH), the results indicate that the Split-Merge Clustering performs better than EvolveCluster. However, when we include the prior data segments and use the entirety of the produced clustering solutions, the results significantly decreased as shown, e.g., in Fig. 3 and Table 4.

7.3 Handling concept drift and outliers

By analyzing the results and how EvolveCluster operates, we aim to discuss the characteristics of EvolveCluster in the presence of outliers. If a prominent outlier exists within a segment, it will not be removed directly by EvolveCluster. There are no mechanisms in place in itself that identify or removes them. Instead, if the outlier drastically differs from the other clusters, the splitting procedure of the algorithm is likely to promote the outlier to a singleton cluster. If no similar data objects arrive in future segments after the created singleton cluster, they will be discarded as a part of the clustering procedure. However, what is seen as an outlier in the current segment might not be determined to be one in future segments.

It is harder to determine if the data objects are outliers or if the streams' concepts are evolving as the stream evolves. In our second experiment, where we generated data using an RBFGenerator, we investigated how EvolveCluster models data streams with different concept drifts. We included constant incremental changes to each cluster present in the data stream, and EvolveCluster managed to model the moving clusters accurately. This type of concept drift is not always easy to identify with incremental approaches, as data is either removed or weighed down as time moves along. With EvolveCluster, we can compare the produced clustering solutions of each segment to identify if a cluster has moved significantly compared to previous segments. If more significant changes are present, it might be sufficient to compare two neighboring clustering solutions, but it is possible to compare segments further apart with drifts that appear slowly over time.

Additionally, to fully capture the dynamics of evolving data streams, we included creation and deletion of clusters in our experiments. These represent the sudden and gradual concept shift scenarios, in addition to when cluster centroids reach the boundaries and suddenly change direction. From the results, we noted that the splitting functionality of EvolveCluster generally manages to model the stream when new clusters emerge. Depending on the current status of the clustering solutions when new clusters emerge, EvolveCluster identifies the need for a split to model more accurately. However, as no specific merge criterion exists and EvolveCluster instead relies on the transitions to merge clusters, there are occasions where both over- and under-clustering occurs. When the clustering solution produced by EvolveCluster represents a single cluster by multiple smaller clusters, it appears to disrupt the efficacy of the splitting procedure on another cluster.

Specifically when the cluster to split and the ones to merge are very close to each other. These tendencies lead us to believe a merge criterion might be necessary to fully capture the dynamics of an evolving data stream, but it will add to the complexity of EvolveCluster.

8 Conclusions

This study has introduced a novel evolutionary clustering algorithm (EvolveCluster) capable of modeling data streams containing evolving data. Specifically, our experiments have shown that the proposed algorithm can retain previous trends and identify new behaviors as they emerge. Compared to similar approaches, EvolveCluster does not require each data segment to be clustered in advance, and it easily identifies the correlation of clusters between segments. Each segment in the data stream is clustered based on how the data was behaving in the previous segment, which produces an efficient clustering.

We have compared our evolutionary clustering algorithm against two similar approaches, namely the PivotBiCluster and Split-Merge Evolutionary Clustering algorithms. The results have shown that the proposed algorithm can cluster a continuous behavior over multiple data segments and identify new clusters as they emerge. Furthermore, the experiments have also revealed shortcomings of the other two algorithms where they tend to alter the given clustering solutions for the worse.

Our future work includes developing and evaluating a distributed version of the proposed evolutionary clustering algorithm. We will also investigate the possibility of a dynamic split criterion to eliminate the need for tuning a parameter. Furthermore, incorporating an additional option to remember cluster centers from data segments earlier than the previous data segment could be an exciting area of research. Adding this option could make it easier for EvolveCluster to identify recurring concepts.

Appendix

See Figs 10 and 11.

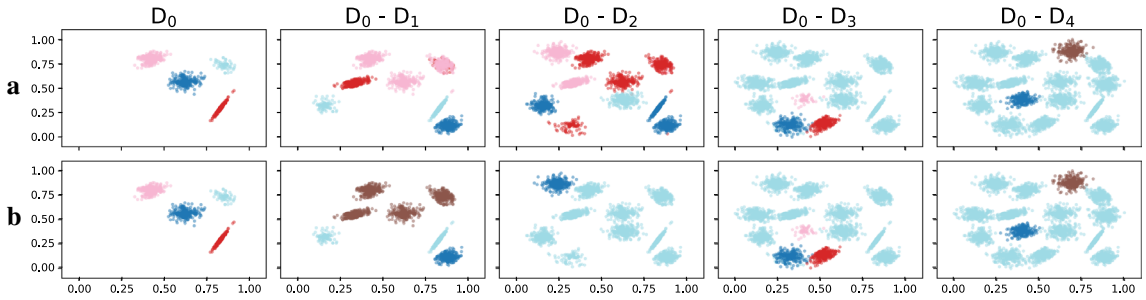

Fig. 10 The clustering solutions obtained on each data segment $D_0, D_1, \ldots, D_4$ of the original S1 dataset, where the all previous segments are included. The results generated by the three studied algorithms are depicted as follows: (a) EvolveCluster, (b) split-merge clustering, and (c) PivotBiCluster. Each color of the data points in the figures represents a single cluster within that segment

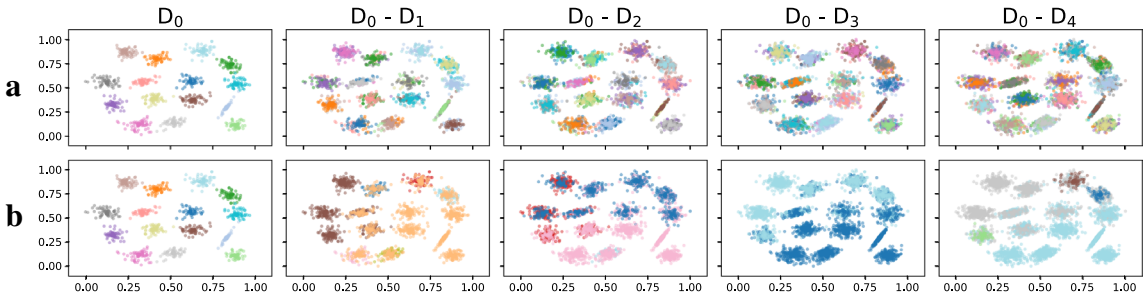

Fig. 11 The clustering solutions obtained on each data segment $D_0, D_1, \ldots, D_4$ of the continuous S1 dataset, where the all previous segments are included. The results generated by the three studied algorithms are depicted as follows: (a) EvolveCluster, (b) split-merge clustering, and (c) PivotBiCluster. Each color of the data points in the figures represents a single cluster within that segment

Acknowledgements We would like to give our special thanks to Milena Angelova, one of the authors in (Boeva et al. 2019), for providing us with their source code to their algorithm and experiments. This work is funded in part by the research project "Scalable resource-efficient systems for big data analytics" funded by the Knowledge Foundation (grant: 20140032) in Sweden.

Funding Open access funding provided by Blekinge Institute of Technology.

References

Aaron B, Tamir DE, Rishe ND, Kandel A (2014) Dynamic incremental k-means clustering. In: 2014 international conference on computational science and computational intelligence, vol 1, IEEE, pp 308–313

Ailon N, Avigdor-Elgrabli N, Liberty E, Van Zuylen A (2012) Improved approximation algorithms for bipartite correlation clustering. SIAM J Comput 41(5):1110–1121

Amit N (2004) The bicluster graph editing problem. PhD thesis, Citeseer

Anderson R, Koh YS (2015) Streamxm: an adaptive partitional clustering solution for evolving data streams. International conference on big data analytics and knowledge discovery. Springer, Cham, pp 270–282

Angelov P, Zhou X (2008) On line learning fuzzy rule-based system structure from data streams. In: 2008 IEEE international conference on fuzzy systems (IEEE World Congress on Computational Intelligence), IEEE, pp 915–922

Arthur D, Vassilvitskii S (2006) k-means++: the advantages of careful seeding. Tech. rep, Stanford

Barddal JP, Gomes HM, Enembreck F (2015) Sncstream: a social network-based data stream clustering algorithm. In: Proceedings of the 30th annual ACM symposium on applied computing, pp 935–940

Barddal JP, Gomes HM, Enembreck F, Barthès JP (2016) Sncstream+: extending a high quality true anytime data stream clustering algorithm. Inf Syst 62:60–73

Baya AE, Granitto PM (2013) How many clusters: a validation index for arbitrary-shaped clusters. IEEE/ACM Trans Comput Biol Bioinform 10(2):401–414

Berkhin P (2006) A survey of clustering data mining techniques. Grouping multidimensional data. Springer, Berlin, pp 25–71

Bifet A, Holmes G, Kirkby R, Pfahringer B (2010a) MOA: massive online analysis. J Mach Learn Res 11:1601–1604

Bifet A, Holmes G, Pfahringer B, Kranen P, Kremer H, Jansen T, Seidl T (2010b) Moa: massive online analysis, a framework for stream classification and clustering. In: Proceedings of the first workshop on applications of pattern analysis, PMLR, pp 44–50

Boeva V, Nordahl C (2019) Modeling evolving user behavior via sequentialclustering. In: Second international workshop on knowledge discovery and user modeling for smart cities (UMCit). Joint european conference on machine learning and knowledge discovery in databases, Springer, pp 12–20

Boeva V, Angelova M, Devagiri VM, Tsiporkova E (2019) Bipartite split-merge evolutionary clustering. International conference on agents and artificial intelligence. Springer, Cham, pp 204–223

Cao F, Estert M, Qian W, Zhou A (2006) Density-based clustering over an evolving data stream with noise. In: Proceedings of the 2006 SIAM international conference on data mining, SIAM, pp 328–339

Cardoso DO, França FM, Gama J (2017) Wcds: a two-phase weightless neural system for data stream clustering. New Gener Comput 35(4):391–416

Chinchor N (1992) MUC-4 Evaluation Metrics. In: Proceedings of the fourth message understanding conference, pp. 22–29. https://aclanthology.org/M92-1002.pdf

Da Silva LEB, Melton NM, Wunsch DC (2020) Incremental cluster validity indices for online learning of hard partitions: extensions and comparative study. IEEE Access 8:22025–22047

Ester M, Kriegel HP, Sander J, Xu X et al (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. Kdd 96:226–231

Fränti P, Virmajoki O (2006) Iterative shrinking method for clustering problems. Pattern Recognit 39(5):761–765. https://doi.org/10.1016/j.patcog.2005.09.012

Gama J (2010) Knowledge discovery from data streams. CRC Press, Boca Raton

Gama J, Rodrigues PP, Lopes L (2011) Clustering distributed sensor data streams using local processing and reduced communication. Intell Data Anal 15(1):3–28

Gao J, Li J, Zhang Z, Tan PN (2005) An incremental data stream clustering algorithm based on dense units detection. Pacific-asia conference on knowledge discovery and data mining. Springer, Berlin, pp 420–425

Ghesmoune M, Lebbah M, Azzag H (2015) Clustering over data streams based on growing neural gas. Pacific-Asia conference on knowledge discovery and data mining. Springer, Cham, pp 134–145

Guha S, Mishra N (2016) Clustering data streams. Data stream management. Springer, Berlin, pp 169–187

Halkidi M, Batistakis Y, Vazirgiannis M (2001) On clustering validation techniques. J Intell Inf Syst 17(2–3):107–145

Handl J, Knowles J, Kell DB (2005) Computational cluster validation in post-genomic data analysis. Bioinformatics 21(15):3201–3212

Hettich S, Bay S (1999) The uci kdd archive. University of California, department of information and computer science, irvine, ca. http://kdd.ics.uci.edu. Accessed Aug 2020

Jaccard P (1912) The distribution of the flora in the alpine zone. 1. New Phytol 11(2):37–50

Jain AK, Dubes RC (1988) Algorithms for clustering data. Prentice-Hall Inc, Hoboken

Jain AK, Murty MN, Flynn PJ (1999) Data clustering: a review. ACM Comput Surv (CSUR) 31(3):264–323

Khamassi I, Sayed-Mouchaweh M, Hammami M, Ghédira K (2018) Discussion and review on evolving data streams and concept drift adapting. Evol Syst 9(1):1–23

Kremer H, Kranen P, Jansen T, Seidl T, Bifet A, Holmes G, Pfahringer B (2011) An effective evaluation measure for clustering on evolving data streams. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 868–876

Kriegel HP, Kröger P, Ntoutsi I, Zimek A (2011) Density based subspace clustering over dynamic data. International conference on scientific and statistical database management. Springer, Berlin, pp 387–404

Lloyd S (1982) Least squares quantization in pcm. IEEE Trans Inf Theory 28(2):129–137

Lughofer E (2008) Extensions of vector quantization for incremental clustering. Pattern Recognit 41(3):995–1011

Lughofer E (2012) A dynamic split-and-merge approach for evolving cluster models. Evol Syst 3(3):135–151

Lühr S, Lazarescu M (2009) Incremental clustering of dynamic data streams using connectivity based representative points. Data Knowl Eng 68(1):1–27

Mirsky Y, Shapira B, Rokach L, Elovici Y (2015) pcstream: a stream clustering algorithm for dynamically detecting and managing temporal contexts. Pacific-Asia conference on knowledge discovery and data mining. Springer, Cham, pp 119–133

Montiel J, Read J, Bifet A, Abdessalem T (2018) Scikit-multiflow: a multi-output streaming framework. J Mach Learn Res 19(1):2914–2915

Moshtaghi M, Bezdek JC, Erfani SM, Leckie C, Bailey J (2019) Online cluster validity indices for performance monitoring of streaming data clustering. Int J Intell Syst 34(4):541–563

Mousavi M, Bakar AA, Vakilian M (2015) Data stream clustering algorithms: a review. Int J Adv Soft Comput Appl 7(3):13

Nordahl C, Boeva V, Grahn H, Netz MP (2019) Profiling of household residents' electricity consumption behavior using clustering analysis. Int Conf Comput Sci. Springer, Cham, pp 779–786

Ntoutsi I, Zimek A, Palpanas T, Kröger P, Kriegel HP (2012) Density-based projected clustering over high dimensional data streams. In: Proceedings of the 2012 SIAM international conference on data mining, SIAM, pp 987–998

O'callaghan L, Mishra N, Meyerson A, Guha S, Motwani R (2002) Streaming-data algorithms for high-quality clustering. In: Proceedings 18th international conference on data engineering, IEEE, pp 685–694

Rousseeuw PJ (1987) Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J Comput Appl Math 20:53–65

Sakoe H, Chiba S (1978) Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans Acoust Speech Signal Process 26(1):43–49

Schubert E, Rousseeuw PJ (2019) Faster k-medoids clustering: improving the pam, clara, and clarans algorithms. Int Conf Similarity Search Appl. Springer, Cham, pp 171–187

Shirkhorshidi AS, Aghabozorgi S, Wah TY (2015) A comparison study on similarity and dissimilarity measures in clustering continuous data. PLoS ONE 10(12):e0144059

Silva JA, Faria ER, Barros RC, Hruschka ER, Carvalho ACd, Gama J (2013) Data stream clustering: a survey. ACM Comput Surv (CSUR) 46(1):1–31

Tibshirani R, Hastie T, Narasimhan B, Chu G (2002) Diagnosis of multiple cancer types by shrunken centroids of gene expression. Proc Natl Acad Sci 99(10):6567–6572

Toussaint W (2019) Domestic electrical load metering, hourly data 1994-2014. Version 1. https://www.datafirst.uct.ac.za/dataportal/index.php/catalog/759. Accessed Aug 2020

Vendramin L, Campello R, Hruschka E (2010) Relative clustering validity criteria: a comparative overview. Stat Anal Data Min 3:209–235

Vinod HD (1969) Integer programming and the theory of grouping. J Am Stat Assoc 64(326):506–519

Wadewale K, Desai S (2015) Survey on method of drift detection and classification for time varying data set. Int Res J Eng Technol 2(9):709–713

Wang X, Mueen A, Ding H, Trajcevski G, Scheuermann P, Keogh E (2013) Experimental comparison of representation methods and distance measures for time series data. Data Min Knowl Discov 26(2):275–309

Zhou A, Cao F, Qian W, Jin C (2008) Tracking clusters in evolving data streams over sliding windows. Knowl Inf Syst 15(2):181–214

Zubaroglu A, Atalay V (2021) Data stream clustering: a review. Artif Intell Rev 54:1201–1236