IJWR

DynamicCluStream: An algorithm Based on CluStream to Improve Clustering Quality

Sahar Ahsani[a], Morteza Yousef-Sanati*[b], Muharram Mansoorizadeh[c]
Department of Computer Engineering, Faculty of Engineering, Bu-Ali Sina University, Iran; s.ahsani@eng.basu.ac.ir[a], mysanati@basu.ac.ir[b], mansoorm@basu.ac.ir[c]

ABSTRACT

Data streams are continuous flows of data objects generated at high rates, requiring real-time processing in a single pass. Clustering algorithms play a vital role in analyzing data streams by grouping similar data samples. Among various time windows for evolving streams, the sliding window method gradually moves over the data, focusing on the most recent information and improving clustering accuracy while reducing memory requirements. The development of distributed computing frameworks like Apache Spark has addressed the limitations of traditional tools in processing big data, including data streams. This paper presents the DynamicCluStream algorithm, an enhancement over Spark-CluStream, which employs a two-phase clustering approach with precise clustering of recent data. The algorithm dynamically determines the number of clusters by merging overlapping clusters during the offline phase, resulting in significant improvements in cluster precision. Experimental results show that it performs up to 47 percent better on average in terms of precision on the CoverType dataset and up to 92 percent better on average in terms of precision on the PowerSupply dataset. Although the algorithm is slower due to data sample removal and cluster integration, its impact is negligible in a distributed environment.

Keywords*— Data mining, Online Clustering, Dynamic Clustering, Stream Clustering, CluStream, Dynamic CluStream.*

1. Introduction

A data stream is a continuous flow of data objects that are generated at different intervals in large quantities and at a high rate, unlike traditional datasets that are stored in a warehouse. Analyzing data streams is crucial and beneficial in various real-world applications, including customer behavior analysis, weather forecasting, stock exchange, urban traffic monitoring, and earthquake prediction [1]. Given the unique nature of a data stream, the available data can only be observed once and must be processed in real-time in a single pass [1, 2].

Clustering is a popular unsupervised learning technique that divides data samples into relatively homogeneous groups based on maximum data similarity within each group and minimum similarity between members of different groups. Data stream clustering algorithms receive the input data using a temporal window and process a part of it that fits within the window. There is a wide variety of time windows for evolving streams, including landmark, fading, tilted, and sliding windows as the most common instances [2-5]. Compared to the other window types, the sliding window method gradually moves over the data and disregards instances that have already been inspected, focusing solely on the most recent data. This approach enhances the accuracy of the clustering algorithm and reduces the amount of memory needed for its operation [6].

Most of data stream clustering algorithms consist of online and offline phases. One of the most popular of which is Clustream [7]. The Clustream algorithm, which is based on partitioning, operates in two phases. In the online phase, summaries of the data stream are stored as micro-clusters. In the offline phase, it performs a k-means clustering algorithm over the generated summaries to obtain *k* user-specified macro-clusters [5, 8]. In general, the

http://dx.doi.org/10.22133/ijwr.2024.417666.1185
Citation S. Ahsani, M. Yousef-Sanati, M. Mansoorizadehc, " DynamicCluStream: An algorithm Based on CluStream to Improve Clustering Quality ", *International Journal of Web Research*, vol.6, no.2,pp.77-87, 2023, doi: http://dx.doi.org/10.22133/ijwr.2024.417666.1185.

*Coressponding Author
Article History: Received: 23 September 2023 ; Revised: 9 December 2023; Accepted: 27 December 2023.

Clustream algorithm serves as a fundamental benchmark often used in comparison scenarios. It's evident that the vast majority of algorithms use the concepts introduced by BIRCH [9], Clustream, or Denstream [10].

The development of distributed computing frameworks such as Hadoop has been driven by the limitations of traditional tools and techniques in effectively processing and extracting information from big data, particularly data streams. The demand for real-time data analysis has led to the development of powerful processing engines like Apache Spark, which builds on the MapReduce model used in Hadoop and can handle various computations like interactive queries and data stream processing. Compared to Hadoop, Spark is faster because it uses main memory for data processing and maintenance, which increases program processing speed [6, 11].

The paper introduces the DynamicCluStream algorithm, emphasizing scalability and precision. Our method represents an enhancement over Spark-Clustream [12], which serves as a distributed version of Clustream. DynamicCluStream employs a two-phase approach similar to basic CluStream, but with more precise clustering of the recent data. During the online phase the algorithm adjusts the window type. Instead of the tilted window that is used in original algorithm, our proposed algorithm utilizes a sliding window mechanism to locally process data. In the sliding window, statistical summary of data is extracted from $[T_c - R + 1: T_c]$, where T_c is current time and R represents the size of the sliding window. Additionally, unlike the original algorithm that determine *k* user specific clusters with k-means algorithm, our method merges overlapping clusters using the common radius concept during the offline phase and dynamically determines the number of clusters. The degree of overlap between pairs of micro-clusters is measured based on their RMSD distances, and sufficiently close micro-clusters are merged. This approach ensures that clusters are combined based on their proximity. This results in 40-90% improvement in cluster precision in certain time intervals. Although the algorithm is slower due to data sample removal and cluster integration, this is hardly noticeable in a distributed environment.

In summary, the main contributions of this research are as follows:

1. The use of sliding windows in the online phase ensures that only data within the window is processed, leading to more concentrated clusters.
2. The proposed method eliminates the need to predetermine number of clusters in the offline phase. This approach prevents merging clusters with different labels and results in higher-quality clusters.
3. The final clustering in the offline phase is performed based on identifying the nearest neighbor for each micro-cluster. If the overlap criterion is satisfied, two clusters are merged.
4. The proposed approach is implemented in the Apache Spark distributed computing framework.

The structure of the paper is as follows: Section 2 reviews the literature on data stream clustering. The functioning of CluStream is explained in Section 3. Section 4 provides a comprehensive overview of the proposed algorithm. In Section 5, the results of an experimental study and evaluation are presented, and the final section summarizes the conclusions.

2. Previous Work

In this section, a review of several data stream clustering algorithms including CluStream is presented. Furthermore, the CluStream is explored in more details, since it has numerous modifications and adaptations in recent years.

One of the recent works in this area is the DstreamEPK discussed by Zhang et al. [13]. The algorithm improves upon CluStream and K-Means for clustering a stream of data. The study analyzed a significant amount of data on users' electricity consumption, with the aim of optimally allocating power resources based on an analysis of users' needs, behaviors, and habits through clustering. DstreamEPK employs a two-phase online-offline clustering method following CluStream, and utilizes a time-decay technique to periodically update and remove micro-clusters, reducing the influence of historical data on clustering. In the offline phase, the Canopy algorithm is used to identify the optimal number of initial clusters (*K*) for K-Means. Also, a highly efficient parallel K-Means algorithm is presented. The experimental results indicate that the proposed approach is stable, efficient, and capable of clustering power data with greater accuracy. However, it should be noted that the number of clusters in the offline phase remains fixed.

A clustering algorithm named CCluStream was introduced in [14], which is an improvement on the CluStream algorithm. CCluStream utilizes the parallel implementation in Spark Streaming and enhances the offline phase of CluStream. The offline phase of CCluStream combines the Canopy and K-Means algorithms, where the Canopy algorithm specifies the value of *K* in K-Means resulting in a more efficient K-Means algorithm. The experiments demonstrate that CCluStream is more efficient and

accurate than the original algorithm, but in some cases, the accuracy of the algorithm decreases with an increase in the number of computing nodes.

Hua et al. [15] described the DEGDS clustering model which enhances CluStream by combining the strengths of density-based and grid-based clustering algorithms. This approach enables the algorithm to produce various cluster shapes and detect outliers while taking advantage of the parallel processing capabilities of the Apache Spark framework to enhance clustering efficiency. The study results indicate that the DEGDS algorithm outperforms CluStream in both accuracy and efficiency. However, this approach may not be suitable for high-dimensional data due to its reliance on a grid-based model, and the quality of the clustering may depend on the network granularity. Additionally, the use of the grid model may cause a decrease in execution speed as the number of nodes increases due to its higher memory requirements.

Sayed et al. [16] introduced the SCluStream as an upgraded version of CluStream for identifying clusters using a sliding window. The experiments have shown that the algorithm outperforms the basic algorithm in terms of both accuracy and scalability. However, the paper does not explicitly present a mechanism for removing data, and these modifications were only evaluated during the online phase of the algorithm.

Bagozi et al. [17] presented the multilevel parallelization approach for clustering big data streams which utilizes CluStream within the Apache Spark distributed framework to identify anomalies and manage outliers. This method offers more powerful parallelization and efficient resource utilization and can adjust parallelism at different levels during operation to reduce computational load and execution time. Although the proposed approach has demonstrated scalability and efficiency, it may become less scalable if the buffer size exceeds the maximum number of micro-clusters allowed. Additionally, it's important to note that the method only focuses on the online phase of CluStream.

Huang et al. proposed a GPU-based parallelized version of the CluStream, PaStream[18]. This algorithm enables the parallel implementation of clustering through buffering the data stream. Measuring the distance between micro-clusters, calculating timestamps of them for decision-making on the removal of micro-clusters, and generating macro-clusters are done concurrently in the offline phase. Additionally, PaStream can detect clusters of various shapes. However, the quality of the algorithm gradually decreases as the buffer size increases.

As another extension of Clustream, CluStream-GT [19] is an online clustering algorithm for the field of health. The algorithm clusters patients with evolving time series data. It is more efficient than other clusterings in terms of speed, however, the accuracy is slightly decreased.

The HPStream [20] algorithm has been developed to improve CluStream for clustering high-dimensional data streams. It assumes that every data sample in the algorithm has a weight that decreases over time, indicating the gradual decay of a cluster. Consequently, the micro-clusters contain statistical information on the decay rate of a cluster as well. However, HPStream is based on the idea of K-Means clustering; therefore, the final clustering results have been obtained with a constant number of clusters.

In [21], a stream clustering algorithm referred to as CluStream-Hybrid has been proposed and developed based on the principles of the clustering process available in CluStream. In the offline phase, it uses the K-Means++ [22] rather than the classic K-Means. The results have demonstrated that CluStream-Hybrid performs runtime computations at an appropriate speed, and at the same time maintains the accuracy of high-dimensional data clustering. However, a constant number of clusters have been assumed in the algorithm.

In [23] a density-based stream clustering algorithm called ARD-Stream, has been proposed. In this method, a dynamic radius threshold is introduced for each micro-cluster in the online phase and it creates the final clusters by considering the shared density and the distance between the micro-clusters. This algorithm is implemented in MOA framework and compared with several existing algorithms, which shows the better performance of ARD-Stream in the field of data stream clustering. However, this method assumes a constant speed of concept evolution and concept drift, whereas in practice, data streams show different rates of change.

In [24], a method for clustering short text streams, termed TEDM (Topic-Enhanced Dirichlet Model), is introduced. TEDM leverages the Dirichlet process mixture model and integrates topic modeling to enhance its clustering capabilities. TEDM employs forgetting and merging algorithms to reduce the impact of micro-clusters generated during single-pass clustering. It identifies and reassigns documents within inappropriate clusters to improve topic concentration and clustering accuracy. Results indicate that TEDM surpasses several recent models in short text stream clustering. However, in large-scale corpora, the large data structures within TEDM may impose limitations on its performance.

In [25] incremental clustering method for large-scale mixed data, arriving continuously as data

streams has been proposed. This algorithm employs a k-prototypes algorithm based on the split technique in order to tackle the incremental object, attribute, and class learning spaces at once. So, when necessary, the final distribution of the clusters has to be updated. Results show that the method is scalable and able to upgrade the efficiency of existing k-prototypes methods when dealing with mixed streaming data.

Forresi et al. [26] proposed a streaming approach to schema profiling called DSC+. This method works under the overlapping sliding window paradigm and generates profiles by clustering the extracted schemas using the k-means algorithm. DSC+ follows a two-phase approach: initially, schemas are pre-aggregated into a coreset utilizing a concise data structure known as summaries, after which clusters are derived from these summaries within the coreset. However, the number of clusters, *K*, is fixed.

EmCStream [27] algorithm has been developed for embedding high-dimensional input data into two dimensions. In This algorithm concept drift is particularly addressed by a stream clustering method based on data stream embedding. Compared with popular algorithms such as DenStream and CluStream, it demonstrates better clustering quality. However, this approach maintains a fixed number of final clusters. To address the issue of determining the optimal number of clusters, *K*, another algorithm called NoCStream has been introduced.

In [28] the problem of the evolution of clusters in a sliding window has been focused. SWClustering algorithm proposes a new clustering structure called Exponential Histogram of Cluster Features (EHCF). This method combines the cluster's temporal feature vector with exponential histogram. In fact, the exponential histogram tracks how data within a cluster evolves while the temporal feature vector indicates any change in the distribution of the data in the cluster. EHCF provides a flexible approach for analyzing cluster evolution. Therefore, improving the quality of clustering and reducing the need for computational resources in data stream clustering.

In [29], a data stream clustering algorithm named CSCS is introduced. This method is improved by using the sliding window model via techniques for window aggregation and searching the nearest neighbor. The proposed algorithm generates and maintains a statistical summary of data within a window using sliding window aggregation. This method employs Local Sensitive Hash (LSH) to quickly locate neighbors. Furthermore, it suggests a clustering policy to decide whether to add a new summary to existing clusters or conduct clustering on the entire summary. However, this technique utilizes the weighted k-means algorithm for re-clustering, with a fixed value for *K*.

in [30] a new k-means clustering algorithm for data streams, called StreamKM++ has been proposed. This method employs landmark windows for data clustering. It builds a coreset tree for the summaries using a k-means++ [22] seeding approach. However, due to the high cost associated with building such a tree, it is not suitable for use with sliding windows [29].

3. The Clustream

The Clustream algorithm [7] employs both online and phases to cluster data streams. During the online phase, statistical summaries of the data stream are stored as micro-clusters. In the offline phase, these summaries are used to generate final clusters for the user across different time horizons. To capture the significance of data at different points in time, the micro-clusters use snapshots that store statistical summaries at varying levels of granularity, forming pyramidal patterns. Newer data samples are stored at finer granularity while older data samples are stored at coarser granularity.

To store general information about the data stream, CluStream uses a cluster feature vector (CF), which is similar to those used in the BIRCH algorithm. However, CF vectors in CluStream also incorporate parameters to store temporal information about clusters. These vectors are continuously updated as new data enters the system. The cluster feature vector in CluStream is a set consisting of $\{CF2^x, CF1^x, CF2^t, CF1^t, N\}$, where N is the number of data samples in the cluster, $CF1^x$ is the linear sum of the cluster data samples, and $CF2^x$ its Sum of Squares of Data Samples. $CF1^t$ and $CF2^t$ parameters show the sum of time stamps of data samples in each micro-cluster and their sum of squares, respectively. The timestamp information in each cluster enables the calculation of the mean and standard deviation for when data samples corresponding to a particular micro-cluster, according to Equ (1) and (2).

$$mean\ time\ stamp = \frac{CF1^t}{n} \quad (1)$$

$$standard\ deviation = \sqrt{\frac{CF2^t}{n} - (\frac{CF1^t}{n})^2} \quad (2)$$

During the online phase of the algorithm, the initial micro-clusters are generated using the standard K-Means method, where the parameter q = 10*K* specifies the number of micro-clusters and *K* represents the number of final clusters. As the algorithm runs, the micro-clusters are continuously

updated by calculating the Euclidean distance between the center of the micro-cluster and each new data sample. If the data sample falls within the maximum boundary of an existing micro-cluster, it is absorbed into that cluster, otherwise, a new cluster is created for that sample. In CluStream, the number of clusters is maximized in the initial steps, so a constant number of clusters are generated in each time unit. Whenever a new micro-cluster is created, the number of current clusters is reduced by one through either the removal of an older micro-cluster or the merging of the two closest micro-clusters.

The radius of each micro-cluster, as given in Equ (3), is determined based on the root mean square deviation (RMSD) of the data samples in the cluster multiplied by a constant factor t = 2. It is important to note that the calculation of RMSD, as shown in Equ (4), can only be performed on micro-clusters that have absorbed more than one data sample. In the case of a micro-cluster with only one data sample, the radius is determined heuristically based on the distance from the nearest micro-cluster.

$$\text{Radius} = t * \text{RMSD} \tag{3}$$

$$RMSD = \sqrt{\frac{\sum CF2^x}{n} - (\frac{\sum CF1^x}{n})^2} \tag{4}$$

During the offline phase, the micro-clusters obtained from the online phase are utilized along with a modified version of the K-Means algorithm to create final clusters in different time horizons. The final number of clusters is specified by the user based on the constant parameter *K*, and the modified K-Means algorithm is used for the final clustering. In this process, the micro-clusters are weighted by their sizes. A major advantage of the modified K-Means algorithm is that it selects the initial cluster centers in a more optimal way, based on the weights of the micro-clusters. This means that clusters with larger numbers of data samples are given higher weight and are more likely to be selected.

In general, A limitation of the CluStream approach is the constant number of clusters in both online and offline phases, which may not be ideal for evolving data streams with complex input data. Additionally, CluStream uses a tilted time window where expired clusters are not completely removed along the stream, which reduces the accuracy of clustering.

The Spark-CluStream algorithm [12] is a distributed version of the CluStream algorithm. In the offline phase, this algorithm provides a different version of the K-Means algorithm. This version essentially uses a personalized version of the K-Means algorithm in Spark, with the difference that it assigns a weighted allocation to each of the micro-clusters based on the number of data samples they contain. It also takes a large number of samples from the centers of the microclusters, which are considered data samples, to produce better centers based on cluster weights. After forming the initial clusters, the algorithm continues with the weighted K-Means algorithm process.

4. The Proposed Algorithm: DynamicCluStream

This section introduces DynamicCluStream, an enhanced version of CluStream. The subsequent section will present the superior performance of this algorithm when compared to the original version. Like the original algorithm, the proposed algorithm involves online and offline phases, both of which are modified and updated to obtain a better performance.

4.1. Motivation

The proposed algorithm is inspired by Clustream in order to improve the Spark-Clustream algorithm. Essentially, Spark-Clustream is a version of Clustream adapted for distributed systems within the Spark framework. The Clustream considers the number of clusters as fixed. However, a fixed number of clusters may not be suitable to accommodate any changes in data stream. Therefore, in our research, in addition to using the sliding window in the online phase to better adapt to the data stream changes, it dynamically determines the number of final clusters based on the overlap radius.

Implementing an algorithm in a distributed environment is to enhance the efficiency of clustering and its feasibility in the presence of big data. In some cases, the large volume of data, its rapid generation, and the need for quick processing make centralized processing impractical or even unfeasible. By employing a distributed approach, clustering can be performed in a distributed environment by leveraging the computational power of aggregated computers to efficiently and effectively process large-scale data. In this research, representative datasets are used to simulate a relatively realistic data stream to evaluate the algorithm's performance where data is rapidly generated.

4.2. Proposed Method

In the proposed online phase, alterations have been introduced to the structure of micro-clusters, as originally presented in the CluStream. The revised structure replaces the parameters related to cluster timestamps within the cluster feature vector with a new parameter called TL. This parameter, represents the last update time of the micro-cluster and enables efficient identification and elimination of expired micro-clusters. In general, expired data are those

generated within a period shorter than the difference between the current time T_c and temporal window length (R), as shown in Equ (5).

$$T < (T_c - R) \quad (5)$$

Moreover, the number of micro-clusters has not been assumed to be constant in this phase for greater dynamicity in clustering, and new micro-clusters are generated gradually upon requirement in practice. It should be noted, that the proposed algorithm behaves like CluStream once the number of micro-clusters reaches the predetermined maximum.

In fact, window model is designed to identify the most recent input data for processing. Generally, more recent information from the stream better reflects the evolving activities in clusters. Therefore, establishing a window to retain this data for clustering purposes can significantly enhance the algorithm's clustering performance. Unlike other window models, only a small number of studies have focused on clustering algorithms with sliding windows[16, 28, 29, 31]. However, the sliding window concentrates only on data objects within the current focus, disregarding older ones. This approach results in fewer clusters in a window interval. As a result, in addition to improved clustering quality, the computation time and resource usage are reduced [5].

In our method, instead of using a tilted window, a sliding window is employed for data management. In the sliding window model, a statistical summary of data for which the time stamp lies in the range [T_c - R + 1: T_c] is provided, where R represents the size of the sliding window, and T_c is the current timestamp. the sliding window uses only the latest data to update the model, and expired clusters are removed based on the same data. Consequently, the proposed method focuses on recent data, resulting in more accurate clusters. Further details regarding the changes made to the online phase of the proposed method can be found in [31].

As previously mentioned, CluStream's offline phase statically defines the number of clusters. This approach can sometimes lead to the partitioning of a cluster into multiple smaller clusters or the aggregation of multiple clusters into a single one, resulting in a potential degradation of cluster quality. To address this limitation, our algorithm proposes a dynamic approach to determine the number of final clusters by aggregating them based on the overlapping radius. The degree of overlap between micro-clusters is determined by their distance, specifically, two micro-clusters are merged if their distance is smaller than the RMSD of the second micro-cluster. This approach ensures that clusters are combined based on their proximity, leading to more accurate cluster identification. Furthermore, the more data in the K-Means algorithm, the better centers are generated. Therefore, in order to achieve desired results in the basic algorithm (Spark-CluStream), micro-clusters are sampled to increase the data quantity, which in turn incurs additional costs. However, the proposed algorithm addresses this challenge by conducting clustering operations in proportion to the available micro-clusters.

Figure 1 demonstrates the merging of overlapping clusters. In the first line, the micro-clusters are converted into RDDs after creation, to be automatically distributed to different nodes by Spark and enable parallel processing. From lines 2 to 7, the closest neighbor is identified for each micro-cluster, and if the overlap criterion is satisfied, its identifier is returned. Line 8 involves merging the identifiers of overlapping clusters into a single identifier. This operation is performed by traversing the graph with the BFS algorithm on the adjacency matrix. In line 9, clusters with the same identifier are merged. Finally, the number of micro-clusters is returned in line 10, and the final micro-cluster centers are returned in line 11.

5. Results and Descussion

In this section, we present the results of experiments conducted to assess the clustering quality and speed. Furthermore, we examine the

Input: **microClusters: get list of microClusters**

1: create a data frame to process micro-clusters in parallel.

٢: **for all** mc ∈ microClusters **do**

3: find the nearest micro-cluster to mc

4: **if** dist (mc_i, mc_j) ≤ mc_j. rmsd

5: microClusters ← (mc, nearest micro-cluster)

6: **else**

7: microClusters ← (mc, -1)

8: **end if**

9: **end for**

10: using the BFS algorithm, change the overlapping micro-clusters' identifiers to a unique identifier.

11: merge all micro-clusters with the same identifier

12: K ← size of microClusters

13: lCentroids ← microClusters.centroids

Figure. 1. DynamicCluStream in offline phase

performance of DynamicCluStream in comparison to Spark-CluStream using real-world datasets.

5.1. Experimental setup and configuration

The algorithm described above has been implemented within the Apache Spark framework, specifically as a stream in Spark Streaming. All experiments were conducted on a Windows 10 device with a Core i5 processor and 16GB of memory. To evaluate the algorithm's performance, three real-world datasets were utilized: Cover Type[1], Power Supply[2] and KddCup99[3]. These datasets are described in Table 1.

The Cover Type dataset used for predicting different types of forest cover across various regions in the United States. Each sample in the dataset consists of fifty-four features related to forest cover information. The Power Supply dataset representing hourly power supply data from an electricity company in Italy. It includes power readings from two sources: power from the main grid and from other transformed grids. Each sample in the dataset contains two features. The dataset covers a three-year period, from 1995 to 1998, capturing the power supply records. The KDDCup99 dataset is one of the most popular datasets used for testing stream clustering algorithms. This dataset comprises non-uniformly distributed network traffic data records. It consists of 41 attributes, of which 34 are numerical features describing the connection. The class label indicates whether the connection is normal or one of the 22 types of network attacks. In line with convention, this paper utilizes the 10 percent subset of the original dataset, which contains 23 classes, 34 numerical features, and 494,021 instances.

It is worth noting that the data were standardized using the standardization function available in Apache Spark. This step was taken to ensure that the inclusion of data with different scales does not adversely impact the algorithm's accuracy. The initialization of the algorithm used N = 2000 data samples, with a data sample entry rate of N = 1000 per second. The window size was set to R = 4 or R = 6, and the number of micro-clusters was specified as q = 10K.

Table 1: Description of the real datasets

Dataset	*Type*	*Instance*	*Dimension*	*Class*
CoverType	Real	581,012	54	7
PowerSupply	Real	29,938	2	24
KddCup99	Real	4,898,431	41	23

[1] https://archive.ics.uci.edu/ml/datasets/covertype
[2] https://www.cse.fau.edu/~xqzhu/stream.html
[3] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

5.2. Clustering quality

One of the most frequent criteria used for internal evaluation of clustering quality is the sum of squared distances, which has been used in this paper for analysis of the favorability of the proposed algorithm. This index measures the similarity among members within a cluster by computing the Euclidean distance between a data point (X_i) and its closest cluster center (Cx_i). SSQ is calculated using Equ(6).

$$SSQ = \sqrt{\sum_{i=0}^{n} D^2(x_i, Cx_i)} \quad (6)$$

5.3. Clustering speed

Another appropriate criterion used for the evaluation of the algorithm is clustering speed, which can be calculated by measuring the execution time of the algorithm. In other words, the time required for processing a group of data indicates the algorithm speed.

5.4. Comparision with the basic Spark-Clustream

In this section, we compare the proposed algorithm to the original algorithm using the criteria introduced in the previous section. Figures 2,3,4 and 5 depict the clustering quality of Spark-CluStream and DynamicCluStream across sliding windows of different sizes and the number of final clusters. Each algorithm was run four times, and the average sum of squares was calculated to generate the diagrams in each figure. The results demonstrate that the proposed algorithm consistently outperforms Spark-CluStream in terms of clustering performance across different datasets. On average, the proposed algorithm achieved up to a 47% increase in accuracy for the Cover Type dataset and up to a 92% increase for the Power Supply dataset. Figure 5 reveals that the results obtained for the Power Supply dataset are always better than those for the Cover Type dataset. This is because the Power Supply dataset follows a uniform distribution.

The original algorithm aims to generate a fixed number of clusters, *K*, in the offline phase, which leads to the merging of clusters with different labels and a decrease in result quality. In contrast, the proposed method allows clusters to merge in the offline phase when necessary and eliminates the need to generate a predetermined number of clusters. This approach prevents the merging of clusters with different labels.

Figures 6 and 7 present the mean running time of DynamicCluStream and Spark-CluStream for various values of *K*. In both experiments, the time

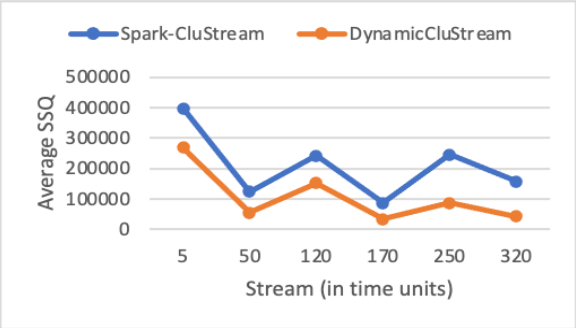

Figure. 2. Results for the CoverType dataset with parameters N = 1000, R = 4, and K = 5

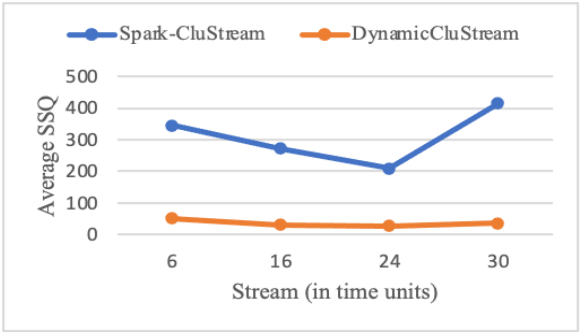

Figure. 5. Results for the Power Supply dataset with parameters N=1000, R=6, and K=10

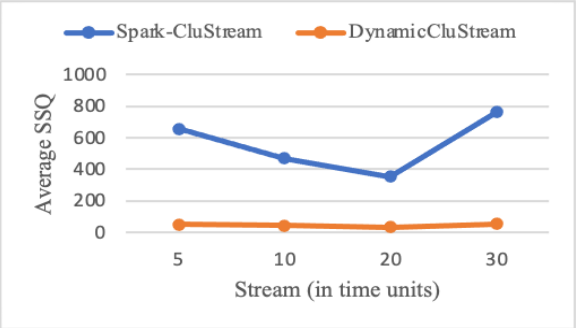

Figure. 3. Results for the Power Supply dataset with parameters N=1000, R= 4, and K = 5

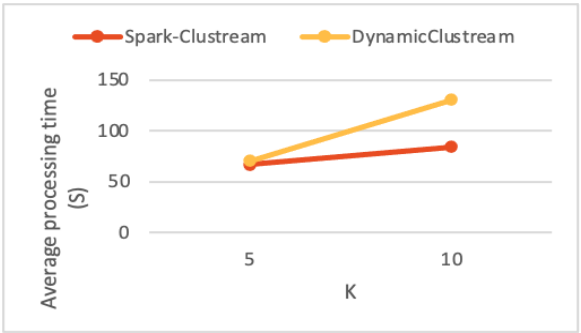

Figure. 6. Processing time for the CoverType dataset

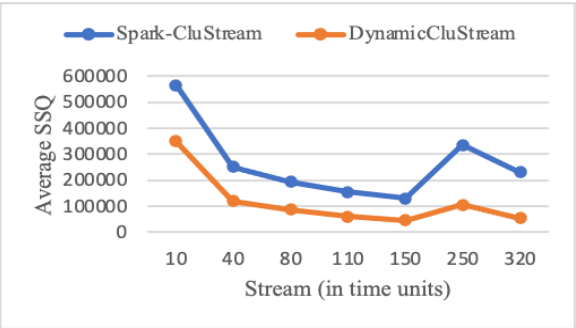

Figure. 4. Results for the Cover Type dataset with parameters N=1000, R= 6, and K = 10

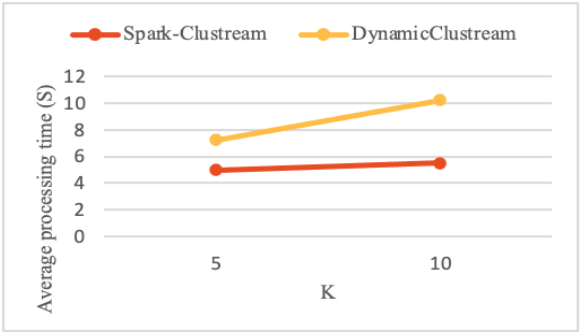

Figure. 7. Data processing time over the Power Supply dataset

spent on algorithm initialization has been ignored. The figures indicate that the execution time of the algorithms for each dataset increases gradually as the value of *K* rises. However, Spark-CluStream exhibits a relatively minor increase in execution time since it generates a fixed number of clusters. On the other hand, the proposed method experiences increased execution time due to the varying number of clusters and the removal of expired clusters from the sliding window. Furthermore, the integration of overlapping clusters in the offline phase has also increased the execution time of the algorithm.

Time complexity of the specifying the number of clusters in the offline phase

The proposed algorithm begins by identifying the closest neighbor for each micro-cluster. If the overlap criterion is satisfied, the algorithm returns the identifier of the closest neighbor. However, if a cluster does not overlap with any others, the algorithm returns a value of -1. With the assumption that the number of micro-clusters is q, the cost of finding overlapping clusters is of O(q2). Next, the identifiers of clusters located within the other's boundaries are changed to a single identifier. This process is accomplished by employing the BFS algorithm on the adjacency matrix representation of the graph. The size of the matrix is determined based on the number of overlapping micro-clusters. Assuming v represents the number of overlapping micro-clusters and graph vertices, the time complexity of traversing the graph is O(v2). In the best-case scenario where no cluster overlaps exist, the processing time and the cost remain constant.

Finally, the micro-cluster centers are returned as the number of clusters. The overall time complexity of the algorithm is O(q2 + v2). However, if the time complexity of traversing the graph is constant, meaning O(1) for v2, the complexity is reduced to O(q2).

Comparison of the proposed algorithm to the original one

The proposed algorithm demonstrates superior performance compared to the original algorithm, which utilizes K-Means in the offline phase, in terms of quality and cost. In terms of quality, the proposed method has performed better than K-Means in most cases.

However, in some time units, the proposed algorithm may not perform well in terms of quality because the length of the window may not be selected properly. In the sliding window model, the result of clustering is strongly influenced by size of the window, so the proper sizing of the window directly impacts clustering quality. Therefore, the clustering precision may decrease, particularly when utilizing a smaller window size. Furthermore, in datasets with a high rate of data change, the length of the window should be chosen more carefully [5].

Regarding cost during the offline phase, it presents lower costs for two primary reasons: firstly, the cost of the proposed method is deterministic, and secondly, it is proportional to the number of micro-clusters rather than the main data.

The deterministic nature of the proposed algorithm means that its execution time over a specific series of input data is always a constant number. Therefore, the variance of its termination time is always zero. On the other hand, K-Means is non-deterministic due to its iterative nature, requiring multiple iterations for convergence. Consequently, it exhibits varying execution times for a specific series of input data, leading to higher costs compared to the proposed algorithm.

As mentioned earlier, the proposed algorithm performs clustering in proportion to the available micro-clusters. In contrast, K-Means benefits from a larger amount of data, leading to improved cluster centers. Therefore, the micro-clusters are sampled to obtain favorable results in the basic algorithm (Spark-CluStream) and thus increase the number of data, which in turn raises the cost. It is important to note that the proposed algorithm occasionally utilizes the BFS algorithm to identify overlapping clusters with identical identifiers. This inclusion increases the time complexity of the proposed algorithm.

Table 2 shows the average percentage improvement of the proposed algorithm (DynamicCluStream) as compared to the basic algorithm (Spark-CluStream) with the internal evaluation criterion (SSQ).

5.5. Comparosion to the other works

In this section, we compared our proposed method with recent data stream clustering algorithms: SWClustering, StreamKM++, CSCS, SClustream, all of which is described in Section 2. To evaluate the clustering quality, we use the Sum of Squared Distance (SSQ) criteria.

Table 3 presents the average SSQ results with various datasets by each method. DynamicClustream has consistently get the best results compared to other algorithms on all three datasets. It accurately assigns data to appropriate clusters, resulting in the highest performance.

All algorithms except StreamKM++ utilize the sliding window model. The StreamKM++ algorithm shows almost the same quality as the CSCS algorithm in the cover type dataset. It appears that the k-means algorithm works well to find good initial centers. However, this operation is time-intensive [29]. The SClustream algorithm provides a lower quality compared to other algorithms in the CoverType dataset. It is probably because this algorithm only reported the clustering results during the online phase. On the kddCup99 dataset, the SWClustering algorithm has provided better results than the rest of the algorithms, however, the proposed method consistently delivers superior results across all the three datasets.

Approximate results of the StreamKM++ algorithm on both the CoverType and KddCup99 datasets, as well as the SWclustering algorithm on

Table 2: Precision improvement of the DynamicClustream over Spark Clustream

Dataset	*Config*	*Spark-Clustream*	*Dynamic-Custream*	*Improved (%)*
CoverType	N=1000, R=4, K=5	237591.1	125205.8	*45%*
PowerSupply	N=1000, R=4, K=5	361.277	33.5373	*92%*
CoverType	N=1000, R=6, K=10	253426.3	138723.4	*47%*
PowerSupply	N=1000, R=6, K=10	839.4457	59.60009	*90%*

Table 3: Average SSQ results of compared methods on real datasets

Method	*CoverType*	*KddCup99*	*PowerSupply*
StreamKM++	≈ 3.47E+09	≈ 2.50E+11	-
SClustream	≈ 3.00E+12	-	-
SWclustering	≈ 9.63E+09	≈ 1.59E+08	-
CSCS	≈ 2.13E+09	≈ 6.67E+10	-
Dynamic-Clustream	**2.38E+05**	**2.91E+05**[a]	**3.61E+02**

[a] *as usual, the experiment is done using the 10 percent subset of the original dataset.*

the CoverType dataset, are obtained from reference [29]. Results for the remaining algorithms are from their respective references.

6. Conclusion

In this paper, a highly efficient algorithm called DynamicCluStream was introduced to cluster big data streams. The proposed algorithm is an improvement over Spark-CluStream, which is implemented within the Apache Spark framework. DynamicCluStream generates clusters from a sliding window and dynamically detects the number of final clusters in the offline phase by aggregating overlapping clusters. This dynamic approach significantly enhances the performance of clustering. In contrast, Spark-CluStream statically specifies the number of final clusters based on the K-Means algorithm, which reduces the quality of the final clusters. Experimental evaluations were conducted on real-world datasets, including coverType, powerSupply, and kddCupp99. These evaluations demonstrated that the proposed algorithm outperforms Spark-CluStream and other algorithms in terms of clustering quality. However, the speed of the proposed algorithm has reduced due to the costly operations of removal and merging.

In future works, we will focus on the improvement of the proposed algorithm for more efficient and timely clustering. Another suggestion involves finding solutions for outlier and noise detection to improve the quality of the induced clusters.

Declarations

Funding

This research did not receive any grant from funding agencies in the public, commercial, or non-profit sectors.

Authors' contributions

During the preparation of this work, the authors used ChatGPT in order to enhance text clarity and writing. After using this tool, the authors reviewed and edited the content as needed. The authors take full responsibility for the content of the publication.

SA: Study design, acquisition of data, interpretation of the results, statistical analysis, drafting the manuscript;

MY: Study design, interpretation of the results, drafting the manuscript, revision of the manuscript, Supervision;

MM: Supervision, drafting the manuscript, revision of the manuscript.

Conflict of interest

The authors declare that no conflicts of interest exist.

References

[1] A. Zubaroğlu and V. Atalay, "Data stream clustering: a review," *Artificial Intelligence Review*, vol. 54, no. 2, pp. 1201-1236, 2021, https://doi.org/10.1007/s10462-020-09874-x.

[2] H. L. Nguyen, Y. K. Woon and W. K. Ng, "A survey on data stream clustering and classification," Knowledge and information systems, vol. 45, pp. 535-569, 2015, https://doi.org/10.1007/s10115-014-0808-1.

[3] M. Carnein and H. Trautmann, "Optimizing data stream representation: An extensive survey on stream clustering algorithms," *Business & Information Systems Engineering*, vol. 61, pp. 277-297, 2019, https://doi.org/10.1007/s12599-019-00576-5.

[4] S. Mansalis, E. Ntoutsi, N. Pelekis and Y. Theodoridis, "An evaluation of data stream clustering algorithms," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 11, no. 4, pp. 167-187, 2018., https://doi.org/10.1002/sam.11380.

[5] X. Wang, Z. Wang, Z. Wu, S. Zhang, X. Shi and L. Lu, "Data Stream Clustering: An In-depth Empirical Study," *Proceedings of the ACM on Management of Data*, vol. 1, no. 2, pp. 1-26, 2023, https://doi.org/10.1145/3589307.

[6] S. Tang, B. He, C. Yu, Y. Li and K. Li, "A survey on spark ecosystem: Big data processing infrastructure, machine learning, and applications," In *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 71-91, Jan. 2022, https://doi.org/10.1109/TKDE.2020.2975652.

[7] C. C. Aggarwal, S. Y. Philip, J. Han and J. Wang, "A framework for clustering evolving data streams," in *Proceedings 2003 VLDB conference*, Elsevier, 2003, https://doi.org/10.1016/B978-012722442-8/50016-1.

[8] F. Ramzan and M. Ayyaz, "A comprehensive review on data stream mining techniques for data classification; and future trends," *EPH-International Journal of Science And Engineering*, vol. 9, no. 3, pp. 1-29, 2023, https://doi.org/10.53555/ephijse.v9i3.201.

[9] T. Zhang, R. Ramakrishnan and M. Livny, "BIRCH: an efficient data clustering method for very large databases," A*CM sigmod record*, vol. 25, no. 2, pp. 103-114, 1996, https://doi.org/10.1145/235968.233324.

[10] F. Cao, M. Estert, W. Qian and A. Zhou, "Density-based clustering over an evolving data stream with noise," in P*roceedings of the 2006 SIAM international conference on data mining*, SIAM, 2006, https://doi.org/10.1137/1.9781611972764.29.

[11] H. Karau, A. Konwinski, P. Wendell and M. Zaharia, *Learning spark: lightning-fast big data analysis*, O'Reilly Media, Inc., 2015.

[12] O. Backhoff and E. Ntoutsi, "Scalable online-offline stream clustering in apache spark," in *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, Barcelona, Spain, IEEE, 2016, pp. 37-44, 1 https://doi.org/0.1109/ICDMW.2016.0014

[13] X. Zhang, Z. Qian, S. Shen, J. Shi, S. Wang, "Streaming massive electric power data analysis based on spark streaming. in *Database Systems for Advanced Applications: DASFAA 2019 International Workshops: BDMS, BDQM, and GDMA, Chiang Mai, Thailand, April 22–25, 2019, Proceedings 24*, Springer, 2019, https://doi.org/10.1007/978-3-030-18590-9_14.

[14] X. Wang and Q. Sun, "Research on Clustream Algorithm Based on Spark," in *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*,

Hangzhou, China, IEEE, 2017, pp. 219-222, https://doi.org/10.1109/ISCID.2017.111.

[15] Z. Hua, T. Du, S. Qu and G. Mou, "A data stream clustering algorithm based on density and extended grid," in *Intelligent Computing Theories and Application: 13th International Conference, ICIC 2017, Liverpool, UK, August 7-10, 2017, Proceedings, Part* II 13, 2017. https://doi.org/10.1007/978-3-319-63312-1_61.

[16] D. Sayed, S. Rady, and M. Aref, "Enhancing CluStream algorithm for CLUSTERING big data streaming over sliding window," in *2020 12th International Conference on Electrical Engineering (ICEENG)*, Cairo, Egypt, IEEE, 2020, pp. 108-114, https://doi.org/10.1109/ICEENG45378.2020.9171705.

[17] A. Bagozi, D. Bianchini and V. De Antonellis, "Multi-level and relevance-based parallel clustering of massive data streams in smart manufacturing," *Information Sciences*, vol. 577, pp. 805-823, 2021, https://doi.org/10.1016/j.ins.2021.08.039.

[18] P. Huang, X. Li and B. Yuan, "A parallel GPU-based approach to clustering very fast data streams," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2015, pp. 23-32,. https://doi.org/10.1145/2806416.2806545.

[19] E. M. Grua, M. Hoogendoorn, I. Malavolta, P. Lago and A. E. Eiben, "Clustream-GT: Online clustering for personalization in the health domain," in *IEEE/WIC/ACM International Conference on Web Intelligence*, 2019, pp. 270-275, https://doi.org/10.1145/3350546.3352529.

[20] C. C. Aggarwal, J. Han, J. Wang, P. S. Yu, "A framework for projected clustering of high dimensional data streams," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, 2004, pp. 852-863, https://doi.org/10.1016/B978-012088469-8/50075-9.

[21] A. Kumar, A. Singh and R. Singh, "An efficient hybrid-clustream algorithm for stream mining," in *2017 13th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*. Jaipur, India, IEEE, 2017, pp. 430-437, https://doi.org/10.1109/SITIS.2017.77.

[22] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Soda*, 2007.

[23] A. Faroughi, R. Boostani, H. Tajalizadeh and R. Javidan, "ARD-Stream: An adaptive radius density-based stream clustering," F*uture Generation Computer Systems*, vol. 149, pp. 416-431, 2023, https://doi.org/10.1016/j.future.2023.07.027.

[24] K. Liu, J. He and Y. Chen, "A topic-enhanced dirichlet model for short text stream clustering," *Neural Computing and Applications*, p. 1-16, 2024, https://doi.org/10.1007/s00521-024-09480-w.

[25] S. Gorrab, F. Ben Rejab, and K. Nouira, "Split incremental clustering algorithm of mixed data stream," *Progress in Artificial Intelligence*, vol. 13, no. 1, pp. 51-64, 2024, https://doi.org/10.1007/s13748-024-00316-1.

[26] C. Forresi, M. Francia, E. Gallinucci and M. Golfarelli, "Dynamic Stream Clustering for Real-Time Schema Profiling with Dsc+", unpublished, Available at SSRN 4701020.

[27] A. Zubaroğlu and V. Atalay, "Online embedding and clustering of evolving data streams," S*tatistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 16, no. 1, pp. 29-44, 2023, https://doi.org/10.1002/sam.11590.

[28] A. Zhou, F. Cao, W. Qian and C. Jin, "Tracking clusters in evolving data streams over sliding windows," *Knowledge and Information Systems*, vol. 15, pp. 181-214, 2008, https://doi.org/10.1007/s10115-007-0070-x.

[29] J. Youn, J. Shim and S. G. Lee, "Efficient data stream clustering with sliding windows based on locality-sensitive hashing," *IEEE Access*, vol. 6, pp. 63757-63776, 2018, https://doi.org/10.1109/ACCESS.2018.2877138.

[30] M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen and C. Sohler, "Streamkm++ a clustering algorithm for data streams," J*ournal of Experimental Algorithmics (JEA)*, vol. 17. pp. 2.1-2.30. 2012, https://doi.org/10.1145/2133803.2184450.

[31] S. Ahsani, M.Y. Sanati and M. Mansoorizadeh, "Improvement of CluStream algorithm using sliding window for the clustering of data streams," in *2021 11th International Conference on Computer Engineering and Knowledge (ICCKE)*, Mashhad, Islamic Republic of Iran, IEEE, 2021, pp. 434-440, https://doi.org/10.1109/ICCKE54056.2021.9721505.

Sahar Ahsani received her MSc in software engineering from Bu-Ali Sina university, Hamedan, Iran In 2021. And her Bachelors Degree in software engineering from Pasargad university, Shiraz, Iran, In 2011. She works at Dotin company as a Data Engineer. Her professional and research interests include big data processing, analysis, and mining, and the latest technologies in this field.

Morteza Yousef Sanati is an assistant professor in the computer engineering department of Bu-Ali Sina University. He received his Ph.D. from McMaster University in 2016 and his MSc (2002) and BSc (2000) from Sharif university of technology, all in software engineering. His main research topics are big data analysis, data mining and blockchain.

Muharram Mansoorizadeh is an associate professor at the Computer Engineering Department of Bu-Ali Sina University. He received his BSc degree in software engineering from the University of Isfahan, Isfahan, Iran, in 2001, and his MSc degree in software engineering and the PhD in computer engineering from Tarbiat Modares University, Tehran, Iran, in 2004 and 2010, respectively. His current research interests include machine learning, affective computing and information retrieval.