Khalilian *et al. Journal of Big Data (2016) 3:1*
DOI 10.1186/s40537-015-0036-x

Journal of Big Data
a SpringerOpen Journal

RESEARCH Open Access

Data stream clustering by divide and conquer approach based on vector model

Madjid Khalilian[1*], Norwati Mustapha[2] and Nasir Sulaiman[2]

*Correspondence:
khalilian@kiau.ac.ir
[1] Islamic Azad University, Karaj Branch, Karaj, Iran
Full list of author information is available at the end of the article

Abstract

Recently, many researchers have focused on data stream processing as an efficient method for extracting knowledge from big data. Data stream clustering is an unsupervised approach that is employed for huge data. The continuous effort on data stream clustering method has one common goal which is to achieve an accurate clustering algorithm. However, there are some issues that are overlooked by the previous works in proposing data stream clustering solutions; (1) clustering dataset including big segments of repetitive data, (2) monitoring clustering structure for ordinal data streams and (3) determining important parameters such as *k* number of exact clusters in stream of data. In this paper, DCSTREAM method is proposed with regard to the mentioned issues to cluster big datasets using the vector model and k-Means divide and conquer approach. Experimental results show that DCSTREAM can achieve superior quality and performance as compare to STREAM and ConStream methods for abrupt and gradual real world datasets. Results show that the usage of batch processing in DCSTREAM and ConStream is time consuming compared to STREAM but it avoids further analysis for detecting outliers and novel micro-clusters.

Keywords: Data mining, Data stream clustering, Vector space model, Divide-and-conquer

Background

Very large databases are required to store massive amounts of data that are continuously inserted and queried. Analyzing big datasets and extracting pattern are valuable in many applications such as web click stream, network traffic monitoring, wireless sensor network etc. For that purpose, there are two groups of techniques for mining huge datasets. One group attempts to solve this problem directly with efficient algorithms. Although these algorithms can achieve acceptable clustering results but insufficiency of data storage capacity leads us to process data dynamically in extracting knowledge. The second group of algorithms refers to streaming data and applies mining techniques [1]. In this kind of process, data have been considered as a stream of data which comes in from one side and exit from the other side and the data is not available to visit and process for the second time. This main property of data stream will be associated with some difficulties in clustering data stream (due to clustering is the focus task in this paper) which include:

1. Due to visiting data once during the processing data in stream, the performance of processing data is crucial.
2. Detecting a change in evolutionary data stream and detecting concept drift during the time whether gradual or abrupt.

Some works have been done in the area of the data stream clustering [2–5]. Despite of many efforts to improve the accuracy of the stream clustering methods [4, 6–16], there are some issues that fail to notice by the previous works in proposing data stream clustering solutions. Some applications such as Intrusion Detection Systems contain many monotonous segments of data in dataset. Monotonous segment is defined as the repetitive data in continuous records e.g. in KDDCUP99 dataset, records between 7820 and 11,488 have very nearly the same value for their attributes (0,1032,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,511,511,0,0,0,0,1,0,0,255,255,1,0,1,0,0,0,0,0,smurf).

Thus, monotonous segment causes many problems such as creation of empty clusters and it is utilizing time for clustering data with no cluster. Due to this fact, clustering of repetitive data in monotonous segments is useless and leads to high computational time for clustering. Furthermore, most researches in data stream clustering have focused on clustering algorithms for numerical data streams [2, 15, 17–20] and there is no reference on monitoring clustering structure for ordinal data streams. In addition, the current data stream clustering algorithms cannot mine true clusters effectively because of their weaknesses in determining important parameters such as k number of exact clusters in stream of data. In other words, it is possible that value of k is changed in stream of data. Considering novelty and concept drift in the stream of data especially in real time era is another important challenge which can be related to determining k number of exact clusters. Indirectly, this weakness (k number of exact clusters) will relate to the problem in finding good scalability in data stream clustering algorithms. In this paper, a novel approach is designed to overcome the difficulties in data stream clustering with consideration on the above mentioned problems to achieve more accurate clustering result in high speed and scalable fashion. The following ideas are adopted in our novel approach which will be extensively described in next sections:

a. A framework of data stream clustering that consists of online and offline components is designed to cluster high scale datasets with ordinal data types and monotonous data in real time processing.
b. Use a vector model and divide-and-conquer to reduce size and complexity of clustering problem.
c. An effective data stream algorithm is proposed for online component.
d. Propose an algorithm to detect concept drift and simplified fading function for removing expired micro-clusters and detect novel micro-clusters and outliers in batch processing.

The remainder of this paper is organized as follows. In "Related work", existing methods for data stream clustering which includes primitive and latest study with respect to their weaknesses and strengths will be reviewed. Background of the proposed method is explained in "Research design and methodology". It also presents the data stream

clustering framework and how the proposed method is employed in the proposed framework and uses for stream clustering. "Results and discussion" and "Conclusion" are dedicated for experimental results of the proposed method and conclusion respectively.

Related work

Clustering is the task of assigning a set of objects into groups (called clusters) so that the objects in the same cluster are more similar to each other than to those in other clusters. Clustering is a main task of explorative data mining and a common technique for statistical data analysis. It was used in many fields including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics. These applications usually store massive amounts of data which cause difficulties to do the task of clustering on entire dataset. Many researchers have focused on data stream as an efficient method to extract knowledge from big data [21–23]. On the other hand, many techniques are employed in data mining area but they must be modified to apply in data stream mining environments. Here, it is important that we note the difference between evolving data and concept drift. Evolving data refers to creating and disappearing objects whereas concept drifts refers to changing clusters in terms of content during the time. Many studies have been executed to support concept drift [24]. Most researchers' interest is to apply techniques for increasing compactness of representation, fast and incremental processing of new data points, clear and fast identification of outliers [25]. It is possible to enumerate two main problems in data stream clustering which are concept change and visiting data once and there are different groups of solutions for data stream clustering from different perspectives that we summarized them in Table 1.

There are also group of methods that look into data processing perspective which can be carried out in two fashions: batch (STREAM [4]) and incremental online (ConStream [27]). In batch processing, each stream is divided to the buckets of data then each bucket

Table 1 General group of methods for data stream clustering

Method	Pros	Cons
Condensation-based [5, 26–29]	Having summary of data (global view) Linear complexity Scalability Additive and subtractive property	Resource constraints Detecting only spherical shape Relearning Applicable in low dimension
Data sampling [4, 30]	Speed up Memory usage Low computational complexity	Low quality
Density –based [9, 15, 31–33]	Arbitrary shaped clusters	Density threshold must be determined Noise sensitivity Outlier sensitivity Applicable in low dimension Relearning
Grid-based [2, 19, 34]	Arbitrary shaped clusters High dimension [14]	Stability Relearning
Hierarchical structure [11, 19, 35, 36]	Support evolving and concept drift No need to determining extra parameters	Relearning Inflexibility

is processed in the memory totally. Despite of batch processing, incremental online methods receive samples from the stream one by one and assign to the nearest cluster.

In this study, STREAM, ConStream and proposed method Divide-and-Conquer STREAM (DCSTREAM) have been compared for efficiency and accuracy in clustering results in data stream clustering. STREAM utilizes Divide-and-Conquer method to overcome difficulties in data stream clustering and should be distinguished from the proposed method. DCSTREAM uses Divide-and-Conquer method based on length of vector as it is described later whereas STREAM divides data by using sampling. ConStream is selected as another comparison method due to its advantages in component-based framework and condensational solution for clustering data stream. Thus, we do not compare the proposed method with the recent stream clustering [15, 37] which are based on density and hierarchical methodology. It is clear that each category of methods can cover only a number of problems and applications.

Research design and methodology

DCSTREAM is the proposed method to cluster huge datasets using data stream clustering which is based on Divide-AND-Conquer k-Means algorithm. The framework of the DCSTREAM is partitioned into two main components which are online and offline according to different functions as shown in Fig. 1. It is noticeable that the online component strongly affects the offline component.

We use window of data for online processing as the size of the window should be determined by the user and depends on the available memory. In this method, data which are active during window time W_i, will be expired during window time W_{i+1}. Consequently, visiting data in whole dataset is possibly once and during the window time only.

Furthermore, data pretreatment in a data stream mining system aims to reformat the original data file to prepare data for clustering. It includes feature selection, feature reduction, data cleaning and data transforming. Although these pretreatment tasks are

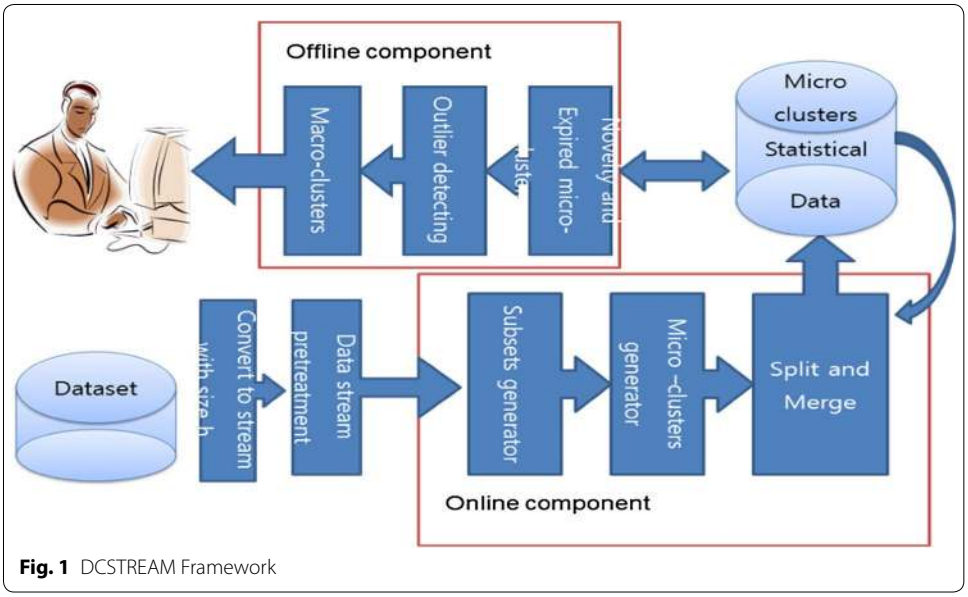

Fig. 1 DCSTREAM Framework

the same for any data stream mining problems, it does not under this study; therefore, we apply general methods in this task whereby feature selection and reduction have been done manually, missing values are ignored and normalizing the data if necessary.

Online component of DCSTREAM

Three main modules are consisted in this component; subsets generator, micro-cluster generator and split and merge for micro-clusters. Data pretreatment module is outside from online component and it is done to preprocess stream data from the original data which is produced by the previous component in the form of data stream. The Divide-AND-Conquer k-Means algorithm has been applied for generating micro-clusters. Concept drift has been managed by frequent split and merge that are stated in the following section.

Subsets generator

This module calculates the length of objects inside window of data and creates one dimension vector for clustering by the *K*-Means algorithm. Based on the result of clustering, the data is divided into some subsets. In fact, objects are categorized into some subsets based on their sizes or levels. Length of objects is defined by $L(O) = \sqrt{\sum_{i=0}^{d} o_i^2}$, d and i are for the number of dimensions and value of its feature respectively.

We utilize average silhouette value to assess the essence of data in terms of monotonous and the best number of subsets, average silhouette value has been defined as the below [38]:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

For each datum i, let $a(i)$ be the average dissimilarity of i with all other data within the same cluster (center of cluster). Then find the average dissimilarity of i with the data of another single cluster. Denote that the lowest average dissimilarity to i of any such cluster by $b(i)$ (centers of all other clusters). If average silhouette value equals 1, the nature of data may monotonous. In addition, we calculate the standard deviation for the entire data in the stream to check whether all the data are of the same value or not. Regardless of being average silhouette value equals 1 for the length of objects in the stream (same size for all objects inside the window), data may be deviated from the mean with respect to all features of the objects and thus it may include some clusters. Therefore, standard deviation is checked (it must be approximately equal to 0). On the other hand, data in the window according to their size may not be included as an intelligent structure that will be obvious as shown by average silhouette value (less than 0.25) [35]. Consequently, this step should be discarded and continued to the next step which is generation of micro-clusters. The details algorithm of subsets generating is shown in Fig. 2.

Micro-clusters generator

For the purpose of achieving greater accuracy in the clustering process, it is necessary to maintain a high level of granularity in the underlying data structures. In order to achieve this goal, we will use a process in which condensed clusters of data points are maintained. We will refer to such groups as *micro-clusters*. In order to generate micro-clusters as shown in Fig. 3, all samples in a window of the stream has been considered for

```
Input:  window of stream data x with size h
Output: subsets of x
1.    for i = 1  to h
2.     calculate length of vectors
3.    end;
4.   k1=2;  initial value for number of subsets
5.   k_means(x1,k1)
6.    s = silhouette(x1); test = mean(s);
7.    if is_nan (test)   % there is no cluster
8.     consider the entire window as the 1 micro-clusters and save statistic into  DB
9.     go to next window of the stream
10.  end;
11.   if (test>=0.99)
12.     generate  2 micro-clusters
13.      if their mean and standard deviation is close to each other
14.         merge  2 micro clusters and save data into DB
15.          go to next window of the stream
16.      else
17.          save data for 2 micro clusters  into DB
18.       end;
19.   if (test<0.99) && (test>=0.25)
20.       find the best value for  the number of subsets based on
21.       average silhouette value
22.        divide vectors based on their clusters into subsets
23.     end;
24.     if  test< .025   % there is no intelligent structure
25.         ignore dividing data into subsets and do normal clustering
26.      end ;
```

Fig. 2 Subsets Generating Algorithm

```
Input: Data subsets inside the window
Output: Micro-clusters
1.    for i =1 to n the number of subsets
2.       find the best value for k number of Micro-clusters inside the subset based on average
     silhouette value
3.        generate Micro-clusters
4.        if test for some Micro-clusters is less than 0.25
5.           re-cluster Micro-clusters
6.        end;
7.     end;
```

Fig. 3 Micro-clusters generating algorithm

clustering called batch processing in opposite of incremental online processing (receiving samples one by one from the stream). According to this strategy, micro-clusters is produced and stored in the repository of micro-clusters for further analysis.

Figure 4 has demonstrated E-R diagram for this repository and attributes which are stored as relations in the repository.

Definition 1: A micro-cluster for a set of data points C at time t is defined as a tuple (*micro-cluster-id, SC2, SC*1*, n, t*). Each record is defined as follows:

Micro-cluster-id: this is an identification to identify every micro-cluster. It is also used to merge, split, concept drift detection, outlier detection and novelty detection.

SC_1: the entry SC_1 contains $\sum_{i=1}^{d} \sum_{j=1}^{d} v_i.v_j$ here, v is a vector with d dimensions and $i \neq j$.

SC_2: the entry SC_2 contains $\sum_{i=1}^{d} v_i$ here v is a vector with d dimensions.

n: the number of elements in each micro-cluster.

t: timestamp of each micro-cluster.

Macro-cluster entity in the repository is utilizing the same definition and attributes. In addition, it uses a weighted value w to apply fading function and decay concept. Every macro-cluster can include more than one micro-cluster and yields a hierarchical structure for clustering.

Split and merge

In order to detect concept drift, a frequent split and merge founded on compactness and separateness criteria has been devised. Basically, when two means of two clusters are approximately close to each other we can merge them in one cluster with respect to their compactness and separateness. Then again, we can split a micro-cluster when the number of elements in the micro cluster becomes too large and its compactness is decreased during the time. Nevertheless, determining parameters as the threshold for split and merge should be done by the user. However, a list of micro-clusters which are registered as frequent split and merge can be used to detect the concept drift. Merge and Split algorithm is shown in Fig. 5.

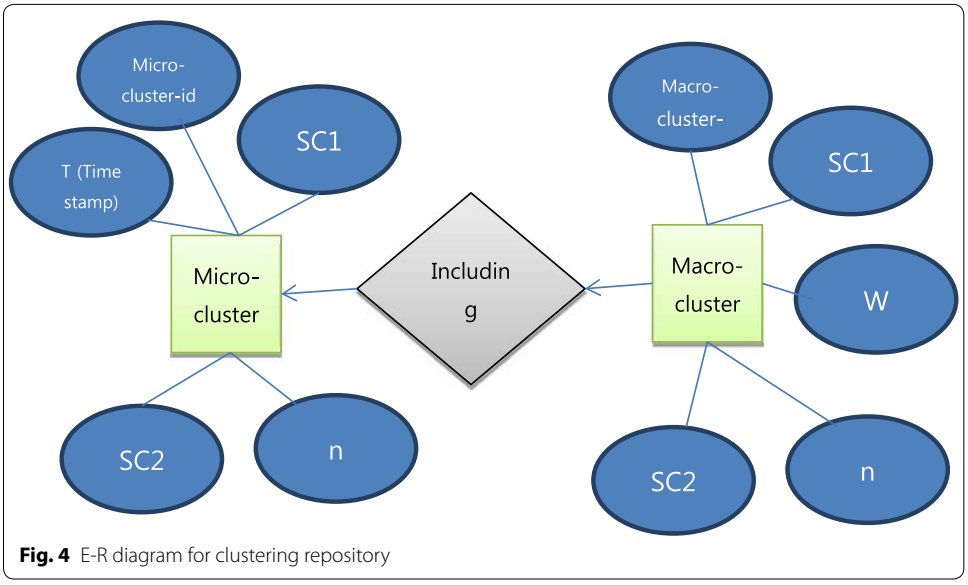

Fig. 4 E-R diagram for clustering repository

```
Input: list of Micro-clusters
Output: Micro-clusters
 for i =1 to n The Number of Micro-clusters
    if 2 Micro –clusters are approximately  equal for average silhouette value
          Merge 2 micro-clusters and update w weight of decay value for all micro-clusters and update DB
    end;
    if there is any Micro-cluster with n number of elements > threshold  and average silhouette <threshold
       split Micro-clusters based on re-clustering
    end;
 end;
```

Fig. 5 Merge and Split algorithm

Offline component of DCSTREAM

This component is designed for further analysis and providing a hierarchical structure for clusters by statistical database that is created by online component. For this purpose, three main tasks have been identified which are macro-cluster generator, outlier detecting, novel and expired micro-clusters. The main task is macro-cluster generator which is simply as same as micro-cluster generator in online component. The only difference is macro-clustering process takes place in offline situation. Since the stream clustering process should provide a greater level of importance to recent clusters, we will provide a time-sensitive weight to each micro-cluster. It is assumed that each micro-cluster has a time-dependent weight defined by the function $w\,(t)$. The function $w\,(t)$ is also referred as the fading function. The fading function $w\,(t)$ is a non-monotonic decreasing function which decays uniformly with time t. In order to formalize this concept, we will define the half-life of a micro-cluster in the repository. The half-life t_0 of a micro-cluster is defined as the time at which $\mathrm{f}\,(t_0) = (1/2)\,f\,(0)$. After each window time, all micro clusters half-life become half except micro clusters which are merged; therefore, half-life of a micro cluster less than a specific threshold is expired.

With the intention of detecting novelty in the data stream, we consider each new micro-cluster which has been registered for the first time in the repository as the novel micro-cluster. In contrast, if this novel micro-cluster does not receive enough data it can be considered as the outlier. To facilitate distinguish between the novel micro-clusters and abnormalities; we will introduce a parameter which is called as outlier factor and defined by the user. This outlier can be a temporal outlier and later by merging other micro-clusters promotes to the real cluster in the next windows. For example, the first document belonging to a particular category in a document stream of current window may be recognized as an outlier, but it may later form a cluster of documents of its own by merging other micro-clusters in the next window. Due to this fact we are employing batch processing for online component, outlier and novel micro-cluster detection can be carried out easily after each time window is in the offline component. Consequently, by determining the time interval for fading function, outlier and novel micro-cluster detection is not required.

We have scrutinized the proposed method DCSTREAM for data stream clustering in an extensive experimental study and the obtained results are discussed. Due to exploit

vector concept in the computational model, it is important that we mention some critical assumptions on using this model as follows:

Our vector space is orthogonal, namely there is no correlation among properties of objects.
Elements of the specific vector have the same data types, meaning that conversion of different data types to one data type for vectors should be done before employing the vector model.
Number of dimensions is equal in all vectors which are entered for processing. In other words, this study does not include pre-processing of data.

Experimental setup

All evaluation tests have been run on a dual processor Intel® Core™ Duo CPU 3.16 GHz with 3.25 GB RAM with, Windows 7 (64bit) operating system. Implementations have been run on MATLAB 8 for coding the DCSTREAM.

Quality metrics

Once the DCSTREAM is implemented, the quality metrics are measured and computed by a set of well defined parameters. There are two main groups of quality metrics including Internal and External evaluation criteria.

Internal evaluation refers to measuring the quality of the clusters without using class label for clusters. These kinds of evaluation usually assign the best score to the algorithm that produces clusters with high similarity within a cluster and low similarity between clusters. One drawback of using internal criteria in cluster evaluation is the high scores on an internal measure do not truly result in data clustering. Additionally, this evaluation is biased towards algorithms that use the same cluster model. For example model used in *k*-Means clustering is naturally optimizes object distances, and a distance-based internal criterion will likely misjudge the resulting clustering. Compactness and Separateness, Silhouette value (combination of compactness and separateness) and SSQ (sum of square distance) can be used to assess the quality clustering algorithms based on internal criterion. In external evaluation, clustering results are evaluated based on the known class labels. These types of evaluation methods measure how close the clustering is to the predetermined benchmark classes. We use two main and popular external evaluation metrics including cluster purity and F-measure criteria for experiments.

Cluster purity measures the percentage of the correct grouped data inside a cluster. Cluster purity is utilized to evaluate the quality of the micro-clusters produced by the online component. Furthermore, cluster purity quantifies a cluster intrinsic coherence. To evaluate the cluster purity, each cluster is assigned to the class which is most frequent in the cluster, and then the accuracy of this assignment is measured by counting the number of correctly assigned samples and dividing by n the number of data in the cluster.

Another well-known evaluation parameter that is considered in the clustering methods specifically for document stream clustering is precision, recall and F-measure. Precision is a number of relevant samples retrieved divided by the total number of grouped samples. On the other hands, precision measures the degree to which the clustering

method produces accurate results. Recall is a number of relevant samples retrieved divided by the total number of samples that actually belong to the real cluster. In addition, recall measures the ability of the clustering method to produce all of the samples that are likely to be grouped by the method. *F-measure* has been defined to balance the contribution of precision (p) and recall (r) according to the following definitions: F-$measure = (2{\cdot}p{\cdot}r)/(p + r)$

Datasets description

For experiments, it is necessary to use some datasets that allows us to analyze effectiveness of proposed method. Our experiments have been conducted on two groups of datasets: synthetic and real world datasets.

Real world datasets We selected the two most popular datasets in this research area. The first dataset is KDDCUP 99 data set which related to the features of network connection data derived from 7 weeks of raw TCP logs consisting of both regular network traffic as well as 24 types of simulated attacks within a military local area network. KDDCUP 99 used in the most cited related works [4, 8, 39]. Each connection is labeled as either normal, or as an attack, with exactly one specific attack type. Each connection record consists of about 100 bytes. The research intends to compare efficiency of DCSTREAM with other available methods in this area. The data is available both as a complete set that contains approximately 4.9 million records and as a 10 % sub-sampled set containing 494,020 points. Each connection record consists of 41 features plus a class ID. Of the available dimensions, 34 continuous valued features were used for clustering and a single outlier point was removed. Accurate clustering of this data demonstrates that the algorithm is able to cope in real world situations where a data stream periodically contains bursts of unexpected and unusual data records. The second dataset is the document datasets which are available in the CLUTO clustering toolkit. It can be obtained from http://www.cs.umn.edu/~cluto. Each document corresponds to a web page listed in the subject hierarchy of Yahoo! (http://www.yahoo.com). The datasets *k1a* and *k1b* contain exactly the same set of documents but they differ in how the documents were assigned to different classes. In particular, *k1a* contains a finer-grain categorization than that contained in *k1b*. We utilize documents obtained from these datasets. This stream contained 163,000 documents. The original datasets includes 20 and 6 classes respectively.

Synthetic datasets We chose PEIVAND data sets to conduct our experiment on synthetic data which can be gathered from PEIVAND website. This website is for finding suitable partners who are very similar from point of personality's view for a person. Based on eight pages of psychiatric questions personality of people for different aspects is extracted. Each group of questions is related to one dimension of personality. Data are organized in a table with 93 columns for attributes of people and 407 rows which are for samples. These dataset is converted into a stream for the testing process. The conversion process is carried out as follows:

- A continuous stream of records is created by concatenating the different instances of the data sets with one another. Since each data set contained $b = 400$ records,

the corresponding stream consisted of 4000 records. The PEYVAND data stream is referred as abrupt. We note that this stream has a very high level of temporal locality in its behavior.

- A second stream is generated from the same set of records, but in this case, the order of the records is randomized. Thus, a data point at a given stage of the stream could be generated from any of the sets of data. We refer to this stream as non-evolving. This stream has almost no temporal locality.
- A third stream is produced which constantly evolves over time. In order to make this smoothly evolving data stream, we applied a block mixing procedure in a sequential fashion. In the first step, the first $2 \cdot b$ records are randomized. In the next step, the block of records in the range $(b, 3 \cdot b)$ are randomized. This process is repeated successively for each contiguous block of $2 \cdot b$ records, at various intervals of b records. The outcome is a data stream in which the evolution is more incessant than the original data. This stream reveals a medium level of temporal locality. We refer to this data set as gradual.

Results and discussion

With the purpose of evaluating the effectiveness of DCSTREAM, we addressed the following questions:

1. What is the best number of subsets for DCSTREAM?
2. How are the quality and the applicability of DCSTREAM on different real world datasets?
3. How does DCSTREAM react to evolutionary data and concept drift in available amount of memory?
4. How is the quality of DCSTREAM in detecting novel micro-clusters and outliers?
5. How is the runtime performance of DCSTREAM?
6. How is the scalability performance of DCSTREAM?

We have considered six experiments below to answer all the above questions and we chose two famous data stream clustering methods STREAM and ConStream in all experiments as the comparison methods.

Experiment 1: Number of subsets analysis

One of the most important parameters which may significantly impact the clustering quality and speed up is the number of subsets in each window. Finding the best value for number of subsets affects on time complexity and results in increased efficiency. Experiment 1 has been carried out to analyze the findings of selecting the best value for number of subsets. It is also demonstrated the effect of choosing the best value for the number of subsets. In this experiment, we find this value by applying k-means algorithm five times to avoid local minima in each window for length of vectors for 100,000 samples and calculating average silhouette for different value of k from 2 to 10. The results of these 9 experiments have been organized in Fig. 6. Each experiment for specific k the number of subsets for windows during the stream progression is shown with a base line. For instance, consider $k = 2$ that is shown in the first base line. It is almost matched the

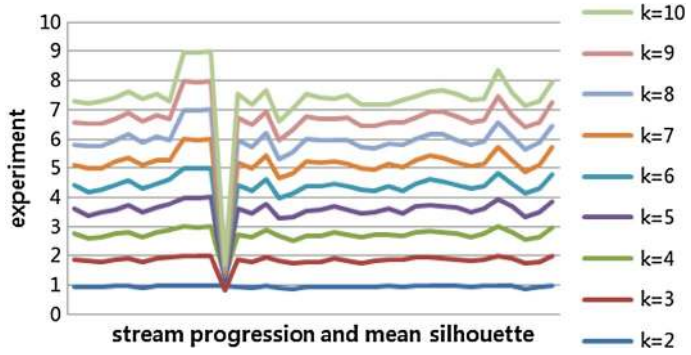

Fig. 6 The average silhouette value for k number of subsets in each window

base line meaning that average silhouette value is 1 or nearly 1. Conversely, for the last experiment with $k = 10$, the average silhouette value is under of the base line meaning that the value is low or even for some window is negative for average silhouette value. Thus, if the result is closer to base line it implies the better outcome otherwise the result is worse. As demonstrated in Fig. 6, there is the most stable value for average silhouette in $k = 2$. Furthermore, null value or value close to 1 (0.999) for mean(s) imply that the entire data in window are at same level and window of data can be considered as one cluster. It is also clear from Fig. 6 that average silhouette value for $k > 2$ in some stream is negative especially when it is closer to $k = 9$ which is the worst value for k number of subsets. Therefore, the number of subsets in each window was set to value 2 for all experiments for KDDCUP 99.

Experiment 2: DCSTREAM quality for real world datasets

We conducted Experiment 2 for measuring the quality and performance of DCSTREAM compared to the two previous methods STREAM and ConStream against the real world datasets. Intrusion Detection Dataset (KDDCUP 99) is chosen as the abrupt instance and document stream as gradual datasets.

One novel feature of DCSTREAM is that it creates a set of micro-clusters for each data window by considering both novelty and outliers. We show the effectiveness and high quality of method in detecting network intrusions. We also compare the clustering quality of our method with STREAM and ConStream using the KDDCUP 99 dataset. STREAM is selected to compare with DCSTREAM because it exploits k-means for clustering stream of data. In fact, it is a good example for condensation-based method. Although, it uses batch processing instead of online incremental processing but it does not support evolving data and concept drift. Thus, STREAM does not support detecting the novelty and the expired clusters. STREAM uses divide and conquer approach to cluster stream of data. The main difference between DCSTREAM and STREAM in using Divide and Conquer method for clustering is that STREAM exploits it for segmentation of entire data (divide), finding the intermediate median for each segment (conquer) and clustering of obtained median as the final results (combine the solutions), whereas DCSTREAM divides data in each stream based on vector length and cluster each subset (conquer). In addition, DCSTREAM employs average silhouette value to find k the best number of subsets in each stream while STREAM divides data without any criterion.

Experiment for KDDCUP 99 has shown that the DCSTREAM has substantially higher quality than STREAM. Figure 8 shows the results where stream speed = 1000 which means that the stream window length is 1000. We run each algorithm 5 times and compute their average of purity. As shown in the Fig. 8, DCSTREAM is always better than STREAM. For example, at first window, the purity of DCSTREAM is close to 1, whereas, STREAM achieved only 0.3 for mean(s). Surprisingly, the high clustering quality of DCSTREAM is achieved while using batch processing instead of incremental online processing. As it can be seen from Fig. 7, the behavior of DCSTREAM is as more stable as STREAM. Due to distinguished data monotonous and outliers in first stage of DCSTREM which is dividing data window into some subsets, DCSTREAM achieves a higher quality and established performance. Furthermore, determining number of clusters for STREAM is required as it affects the quality because determination of k value is not deterministic for a piece of data and it varies from one stream to the other streams, whereas DCSTREAM exploits average silhouette value to determine number of micro-clusters in each stream and identify k automatically.

Although, both STREAM and DCSTREAM utilize batch processing for stream clustering but there is no solution for monotonous data in STREAM. Therefore, STREAM attempts to cluster segment of data by which is not required to cluster. As a result, quality of clusters discovered by STREAM decreased especially for streams 7, 8 and 9 which

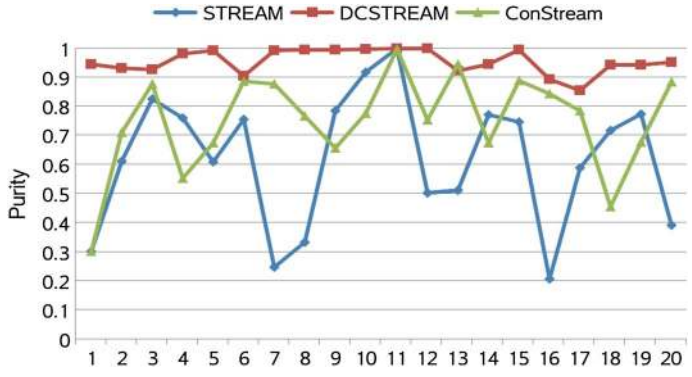

Fig. 7 Quality comparison for STREAM, DSTREAM and ConStream

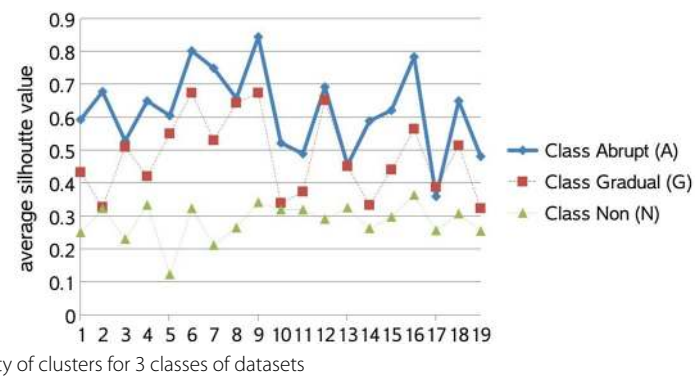

Fig. 8 Quality of clusters for 3 classes of datasets

include monotonous data, while DCSTREAM could detect monotonous data in these streams and ignored clustering them. Furthermore, KDDCUP 99 includes abrupt concept drift that is not recognizable by STREAM. Thus, sudden fall for quality can be seen from Fig. 7 in some streams e.g. stream 13. In streams 16 and 17, the quality decreased for both STREAM and DCSTREAM. In fact, outlier detection is the reason of decreasing quality. Data in these windows are in same class (normal) but the samples include outliers. However, DCSTREAM could achieve better quality than STREAM in this situation. There is a steep fall in quality for the first stream of data clustering by STREAM. First stream is very critical for clustering because of initialization. As it can be seen from Fig. 7 DCSTREAM is more successful than STREAM because DCSTREAM uses the two stages algorithm based on divide-and-conquer to avoid initialization problem of *K*-Means which is used by STREAM.

On the one hand, the divide and conquer enables DCSTREAM to approximate a hierarchical structure based on level of objects as closely as desired. This is in contrast to other clustering algorithms that are only based on the *k*-means data stream clustering with its weaknesses such as initial value for clusters and outlier detection. STREAM achieves a hierarchical structure but it is inflexible structure which is not possible to modify the hierarchical structure.

It is evident from Fig. 7 that even the quality of *ConStream* is better than STREAM but results of DCSTREAM are still higher than *ConStream*. Besides, ConStream employs incremental online processing to create micro-clusters. ConStream used a method for distinguishing outliers and novel micro-clusters by which the quality and performance has been affected. It also causes some other problems that must be solved such as time of using decay concept for outlier and novelty detection. Due to the use of a simple fading function and decay concept with batch processing these problems in ConStream are disappeared for DCSTREAM.

As shown in Fig. 7, there is a sudden fall in the first window because ConStream utilizes *K*-Means for initializing the micro-clusters similar to STREAM. In contrast, DCSTREAM is initialized with acceptable quality for micro-clusters as result of determining *k* number of micro-clusters automatically and employing two stages algorithm. It is obvious from Fig. 8 that, the quality of DCSTREAM has since begun to level out upcoming 1. There is a sharp decline for DCSTREAM in streams 6, 13 and 17. Based on experiment 1, window 13 includes a big diversity in data with a steady changed in samples. ConStream is at a peak in stream 11 similar to STREAM and DCSTREAM, this occurs because of monotonous data in this stream. Since DCSTREAM enables us to avoid clustering for monotonous data and leads to save in time consuming.

In Table 2, we reports the one way ANOVA test for clustering quality of DCSTREAM, ConStream and STREAM methods which has been carried out by IBM SPSS 20. A one way between groups analysis of variance was performed to investigate quality of clustering differences in mean quality obtained by methods. Preliminary assumption testing was conducted to check normality and homogeneity of variance with no serious violations noted. The ANOVA test revealed that there was a statistically significant deference in the mean quality between the three methods. Tukey Post Hoc multiple comparisons test shows that there was a statistically significant difference in the mean clustering quality for the following pairs: DCSTREAM, *ConStream* and STREAM. An inspection mean

Table 2 One way ANOVA test for multiple comparisons

(I) Clustering methods	(J) Clustering methods	Mean difference (I-J)	Std. Error	Sig.	95 % Confidence interval	
					Lower bound	Upper bound
DCSTREAM	*ConStream*	*.20615**	*.05246*	*.001*	*.0799*	*.3324*
	STREAM	*.33713**	*.05246*	*.000*	*.2109*	*.4634*
ConStream	DCSTREAM	−.20615*	.05246	.001	−.3324	−.0799
	STREAM	*.13098**	*.05246*	*.040*	*.0048*	*.2572*
STREAM	DCSTREAM	−.33713*	.05246	.000	−.4634	−.2109
	ConStream	−.13098*	.05246	.040	−.2572	−.0048

Dependent variable: clustering quality

Tukey HSD

* The mean difference is significant at the 0.05 level

quality of the clusters based on post hoc comparisons using Tukey test indicated that DCSTREAM obtained significantly higher mean in clustering quality than *ConStream* and STREAM. *ConStream* also achieved considerable higher mean quality of clustering than STREAM but it is not significant at 0.05 level (sig = 0.04).

To show that DCSTREAM is applicable in different domain we applied DCSTREAM for document stream clustering and compared the method to the *ConStream* which is the recent method for streaming text clustering. We further note that DCSTREAM does not require an input parameter k which defines the number of clusters but this parameter must be determined for ConStream. In order to test the relative effectiveness of the two methods, we used an input parameter k which was equal to the number of natural clusters in the data set for ConStream. For the sake of testing, the entire data stream was clustered in one pass; precision and recall of the two methods were measured.

In Table 3 we have illustrated the precision, recall and F-measure behavior of the DCSTREAM method with respect to the *ConStream*. The precision and recall were computed against the true clusters which were either known from the base data in the text dataset. It is clear that the DCSTREAM substantially outperforms the *ConStream* in terms of both precision and F-measure. In contrast, DCSTREAM outcomes lower recall that *ConStream.* Due to using average silhouette value for detecting outliers and low quality micro-clusters, some samples identify in wrong micro-clusters.

Experiment 3: Evolutionary and concept drift results

Evolutionary and concept drift are two main problems for data stream clustering. In this experiment, we investigated how well the method grouped records corresponding to a given class or category in a cluster for evolutionary and concept drift. For the case of the synthetic datasets, we use an unknown label dataset which was introduced earlier as the seed for creating three classes of data including abrupt, gradual and non-evolving.

Table 3 Quality comparison for document stream clustering

Method	Precision	Recall	*F*-measure
ConStream	44.7	45.1	44.89
DCSTREAM	54.2	43.9	48.5

Additionally, values of features in this dataset are ordinal data type. We note that for unknown label datasets, the value of the silhouette can be computed and used for evaluation. Since we use an unknown label dataset for synthetic dataset, it has been clustered in 4 groups. For this purpose, we choose different number of k for clustering to find the best value for the number of clusters with different clustering method to achieve the highest average value for silhouette. The best number of cluster with the highest average silhouette value was achieved when $k = 2$. Despite of the best value for k, we select $k = 4$ as the second best value for k because generation of three classes for evolutionary test is not possible with only two clusters of data. In contrast, quality of the clusters would be decreased based on silhouette value and it affects the synthetic datasets for further analysis.

We have illustrated the cluster average silhouette value results using the synthetic datasets in Fig. 8. This experiment was conducted for stream length with 1000 points. The quality is totally low using $k = 4$ for seed dataset to create stream of data because of the above mentioned problem. The abrupt class stream is on top with the highest level of cluster silhouette, whereas, the non-evolving stream is on the bottom with the lowest level of average silhouette value. The reason for high quality in abrupt class is that DCSTREAM can detect monotonous segment more than gradual and non-evolving classes. The worst situation is in non-evolving case because there is a big diversity in samples. As a result, it decreased the average silhouette value in non-evolving class.

Experiment 4: Novel micro-clusters and outlier detection

The purpose of this experiment is to test the effect of the evolution process on the clusters and outliers detected by the DCSTREAM. For this reason, we tested the number of new outliers created at the end of window time. In general, outliers were created by substantial changes in the stream behavior. As mentioned, after each window time, novel micro-clusters and outliers are distinguishable. An outlier is identified based on user specified number of micro-cluster members and the average silhouette value. On the other hand, after each stream micro-clusters are merged and split based on defined criteria which were discussed earlier. Novel and expired micro-clusters are recognizable after each split and merge. For all experiments, we assumed a slowly decaying stream in which the half-life was set to half for all micro-clusters except those micro-clusters that take part in split and merge process after each stream. Thus, some micro-clusters may be expired based on determined threshold for the decay value by the user. It can be depended on memory available. By batch processing in each stream, detection of the novel micro-clusters with respect to repository of micro-clusters can be accomplished easily. In Fig. 9, we have a similar problem to Experiment 3. Due to big diversity in data for non-evolving class, many micro-clusters were detected by DCSTREAM. It refers to temporal locality in data. It is clear from Fig. 9 that the least number of micro-clusters were detected in abrupt class. For the case of the gradual and non-evolving data streams, the fraction of new cluster creations was not uniform. However, the average number of new cluster creations was higher in this case because of the greater non-uniformity of the data stream in the gradual and non-evolving datasets.

More details of the quality of the original results are illustrated in the number of outliers in a data stream which is highly dependent upon the ordering and skew of the data

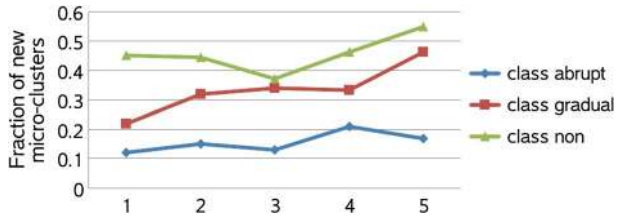

Fig. 9 New micro-clusters detection

stream (Fig. 10). Clearly, when there are bursts of homogeneous behavior within a given temporal locality of the data stream named as monotonous data in this study, very few outliers are observed. The rate of outlier generation is thus an indirect indicator of the level of temporal locality of the data stream. It can also help us to extract knowledge about monotonous segments of data to find normal and abnormal behavior of data.

Experiment 5: DCSTREAM performance

Performance of DCSTREAM can be evaluated in different directions. In this experiment, we test DCSTREAM runtime for KDDCUP 99 as the abrupt dataset against STREAM and ConStream and later we evaluate DCSTREAM against synthetic dataset. We have set the window size as 1000 for all experiments. Figure 11 illustrates the stream processing rate for each of the methods with KDDCUP 99. It is obvious that, our method is significantly faster than the STREAM and ConStream. It is apparent that in each case, several thousand data points per minute could be processed. We also note that the running time was not uniform across the entire execution. This suggests that the approach is extremely efficient and robust for KDDCUP 99 dataset. It is noticeable that each scale in x-axis represents 65000 samples. The main reason of DCSTREAM efficiency is on

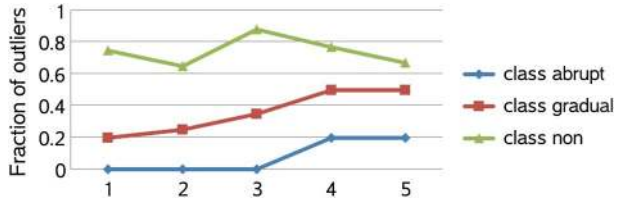

Fig. 10 Outlier detection

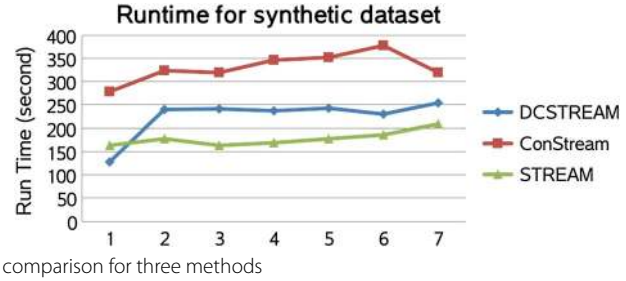

Fig. 11 Runtime comparison for three methods

using two stages algorithm for clustering data by which detect monotonous data segment and avoid extra-clustering process. In other words, DCSTREAM is able to cluster data stream in the first step of the algorithm. It is clear that saving time consumption occurs during the stream clustering.

Despite of ConStream that employs incremental online processing to cluster data, DCSTREAM exploit batch processing with fading function to manage both evolving data and detecting outliers. ConStream uses a method during the clustering to detect novel micro-clusters and outliers which is time consuming (trend setter and mature cluster) because of applying fading function during the stream clustering [27]. In contrast, DCSTREAM detect novel micro-clusters and outliers after each stream processing automatically and uses fading function only for expired micro-clusters to remove from the active memory.

It is obvious from the Fig. 11 that STREAM is the best case for efficiency. Entire data clustering occurs in a hierarchical structure with sampling and divide-and-conquer approach. Thus, evolving data is not taken into account and there is no overhead in runtime of STREAM. The behavior of STREAM is stable because it uses batch processing for all streams with the specific parameters. In first segment of data, DCSTREAM demonstrates better performance because of existing monotonous data.

Experiment 6: DCSTREAM scalability

Several scale-up experiments were performed to investigate how well the execution time is for DCSTREAM scales with respect to portion, dimensions of data and length of data stream. As it is described earlier, it is not required to evaluate DCSTREAM for different number of k because DCSTREAM determines k number of micro-clusters automatically.

In Experiment 6, we investigate DCSTREAM scalability for synthetic datasets in three classes. As shown in Fig. 12, variables dependency between DCSTREAM runtime and portion of data is nearly linear, abrupt, gradual and non-evolving datasets. DCSTREAM runtime for abrupt dataset is little distorted from linear at the end of dataset but in non-evolving it is completely linear.

A near linear relationship between DCSTREAM runtime and number of dimensions was discovered in the synthetic datasets which is presented in Fig. 13.

DCSTREAM runtime is a ratio scale variable which is a positive measurement on a non-linear scale, approximately at exponential scale for the entire length of stream.

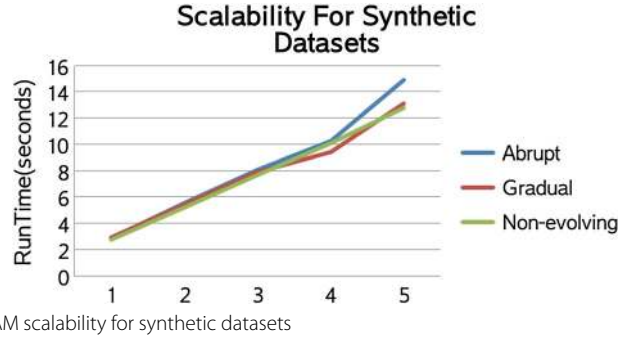

Fig. 12 DCSTREAM scalability for synthetic datasets

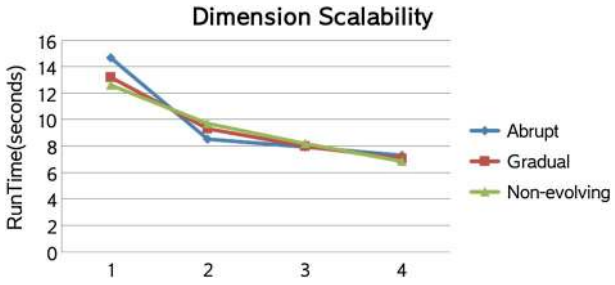

Fig. 13 Dimension scalability for DCSTREAM

The execution time of DCSTREAM with respect to the stream length of data stream is shown in Fig. 14.

An interesting contrast is for stream length 500 and 1000 where runtime in the length of 500 contains less than 1000 points. This exception refers to synthetic data creation methodology where we use blocks with 500 points for stream data.

Conclusion

In this study, a new method based on vector model and divide-and-conquer approach has been utilized for clustering evolving stream data. We have selected a vector model description because it allows us to model the hierarchical structure and relationships among data more accurately by which we are capable to overcome monotonous data and managing ordinal data type. Frequent split and merge provides an effective and detailed representation of any changes that may occur in the cluster over time. A key aspect of DCSTREAM is the use of dividing samples based on their length within the vector model description which allows the method to capture the general structure of the clusters without requiring the complete cluster data to be stored in memory. Most of the time there is no requirement to execute second stage of algorithm. In other words in first stage micro-clusters can be revealed. Thus, algorithm can save memory and processing time. A repository has been used to recall previously discovered micro-clusters in the presence of recurrent change to manage evolving data. For managing evolving data, a fading function with decay concept has been employed by DCSTREAM to detect expired micro-clusters. This enables the algorithm to discard the least useful micro-clusters with

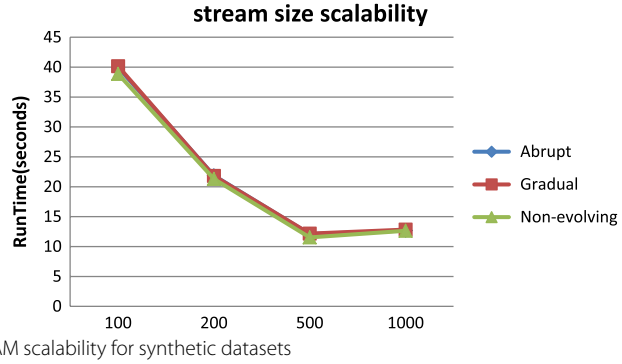

Fig. 14 DCSTREAM scalability for synthetic datasets

memory constraint consideration. The repository can also be used to obtain a historical perspective on the general hierarchical of the micro-clusters over time and to archive these changes for off-line analysis. Experimental results have demonstrated that the algorithm is able to effectively classify both synthetic and real world datasets. The algorithm was compared against implementation of ConStream and STREAM in terms of accuracy and performance using KDDCUP99 and document datasets. The results show that DCSTREAM can achieve superior quality and performance as compare to the mentioned methods for abrupt and gradual real world datasets. Furthermore, quality and performance experiment results for KDDCUP99 demonstrated that monotonous data which was introduced in this study was maintained by DCSTREAM efficiently.

A deep detailed study is required for document stream clustering in different aspects. Time series and transaction applications, uncertain data and fuzzy consideration, arbitrary and non-convex shaped clusters can be considered for the future study.

Authors' contributions

MK is the principal researcher for this study. His contributions include the fundamental idea, literature review, initial drafting of the article, and results implementation. NM guided the initial research idea, and played an essential role in editing the paper. NS also guided the research idea and discussed about the results to write the conclusion. All authors read and approved the final manuscript.

Author details

[1] Islamic Azad University, Karaj Branch, Karaj, Iran. [2] Faculty of Computer Science and Information Technology, UPM University, Serdang, Malaysia.

Acknowledgements

The authors would also like to acknowledge the support provided by Islamic Azad University Karaj Branch.

Competing interests

The authors declare that they have no competing interests.

Received: 28 June 2015 Accepted: 30 November 2015
Published online: 07 January 2016

References

1. Xu R, Wunsch D. Survey of clustering algorithms. IEEE Trans Neural Netw. 2005;16(3):645–78.
2. Tu L, Chen Y. Stream data clustering based on grid density and attraction. ACM Trans Knowl Discov Data (TKDD). 2009;3(3):12.
3. Aggarwal CC. A Framework for Clustering Massive-Domain Data Streams, presented at ICDE '09. IEEE 25th International Conference on Data Engineering; 2009.
4. Guha S, Meyerson A, Mishra N, Motwani R, O'Callaghan L. Clustering data streams: theory and practice. IEEE Trans Knowl Data Eng. 2003;15:515–28.
5. Aggarwal CC, Yu PS. A framework for clustering massive text and categorical data streams. In: SDM; 2006.
6. Zhang, Ramakrishnan, Livny. BIRCH: an efficient data clustering method for very large databases. Presented at ACM SIGMOD Conference on Management of Data; 1996.
7. Yunyue Z, Dennis S. StatStream: statistical monitoring of thousands of data streams in real time. In: Proceedings of the 28th international conference on Very Large Data Bases. Hong Kong, China: VLDB Endowment; 2002.
8. Aggarwal C, Jiawei H, Jianyong W, Philip SY. A framework for clustering evolving data streams. In: Proceedings of the 29th international conference on Very large data bases—Volume 29. Berlin, Germany: VLDB Endowment; 2003.
9. Chen Y, Li T. Density-based clustering for real-time stream data. In: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM; 2007.
10. Cormode, G, Muthukrishnan, S, Wei Z. Conquering the Divide: Continuous Clustering of Distributed Data Streams. In: IEEE 23rd International Conference on Data Engineering, 2007. ICDE 2007, p. 1036–45.
11. Rodrigues PP, Gama J, Pedroso JP. Hierarchical clustering of time-series data streams. In: IEEE transactions on knowledge and data engineering; 2007, p. 615–27.
12. Aoying Z, Feng C, Ying Y, Chaofeng S, Xiaofeng H. Distributed data stream clustering: a fast EM-based approach. Presented at IEEE 23rd International Conference on Data Engineering, 2007. ICDE 2007.
13. Aggarwal CC. On high dimensional projected clustering of uncertain data streams. Presented at IEEE 25th International Conference on Data Engineering, 2009. ICDE '09.
14. Chen Z, He R, Li Y. Online fractal dimensionality reduction in time decaying stream environment. In: Eighth international conference on fuzzy systems and knowledge discovery (FSKD), vol 3. IEEE; 2011. p. 1480–4.
15. Tu Q, Lu JF, Yuan B, Tang JB, Yang JY. Density-based hierarchical clustering for streaming data. Pattern Recognit Lett. 2012;33(5):641–5.

16. Guha S, Meyerson A, Mishra N, Motwani R, O'Callaghan L. Streaming-data algorithms for high-quality clustering. Presented at Proceedings 18th International Conference on Data Engineering; 2002.
17. Aggarwal CC. On High Dimensional Projected Clustering of Uncertain Data Streams. Presented at IEEE 25th International Conference on Data Engineering, ICDE '09.
18. Vivekanandan P, Nedunchezhian R. Mining data streams with concept drifts using genetic algorithm. Artif Intell Rev. 2011;36:163–78.
19. Pardeshi B, Toshniwal D, Meghanathan N, Kaushik BK, Nagamalai D. Hierarchical clustering of projected data streams using cluster validity index advances in computer science and information technology. vol. 131, Communications in Computer and Information Science, Berlin: Springer; 2011. p. 551–9.
20. Cardoso DD, Lima PM, De Gregorio M, Gama J, França FM. Clustering data streams with weightless neural networks. In: ESANN; 2011.
21. Ikonomovska E, Loskovska S, Gjorgjevik D. A survey of stream data mining. In: Proceedings of the 8th National Conference with International Participation. 2007. pp. 19–25.
22. Gaber MM, Zaslavsky A, Krishnaswamy S. Mining data streams: a review. ACM Sigmod Record. 2005;34(2):18–26.
23. Daniel B. Requirements for clustering data streams. SIGKDD Explor Newsl. 2002;3:23–7.
24. Wang H, Fan W, Yu PS, Han J. Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining. ACM; 2003. pp. 226–35.
25. Barbara D. Requirements for clustering data streams. ACM SIGKDD Explorat Newslett. 2002;3:23–7.
26. Aggarwal CC. A Framework for Clustering Massive-Domain Data Streams. Presented at IEEE 25th International Conference on Data Engineering, ICDE '09.
27. Aggarwal C, Yu P. On clustering massive text and categorical data streams. Knowl Inform Syst. 2009;24:171–96.
28. Aggarwal, CC, Yu PS. A Framework for Clustering Uncertain Data Streams. Presented at IEEE 24th International Conference on Data Engineering, ICDE 2008.
29. Tian Z, Raghu R, Miron L. BIRCH: an efficient data clustering method for very large databases. SIGMOD Rec. 1996;25:103–14.
30. Heinz C, Seeger B. Cluster kernels: resource-aware kernel density estimators over streaming data. IEEE Trans Knowl Data Eng. 2008;20:880–93.
31. Wan L, Ng WK, Dang XH, Yu PS, Zhang K. Density-based clustering of data streams at multiple resolutions. ACM Trans Knowl Discov Data (TKDD). 2009;3(3):14
32. Chehreghani MH, Abolhassani H, Chehreghani MH. Improving density-based methods for hierarchical clustering of web pages. Data Knowl Eng. 2008;67:30–50.
33. Yang D, Rundensteiner EA, Ward MO. Neighbor-based pattern detection for windows over streaming data. In: Proceedings of the 12th international conference on extending database technology: advances in database technology. ACM; 2009. p. 529–40.
34. Lin G, Chen L. A grid and fractal dimension-based data stream clustering algorithm. In: International symposium on information science and engineering, ISISE'08, vol 1. IEEE; 2008. p. 66–70
35. Wei J, Brice P. Data stream clustering and modeling using context-trees. Presented at 6th International Conference on Service Systems and Service Management, ICSSSM '09.
36. Chen K, Liu L. HE-Tree: a framework for detecting changes in clustering structure for categorical data streams. VLDB J. 2009;18:1241–60.
37. Hongbin G, Ruiguang L, Jie H. A Kind of Data Stream Clustering Algorithm Based on Grid and Relative Density. Presented at Spring Congress on Engineering and Technology (S-CET); 2012.
38. Kononenko I, Kukar M. Machin learning and data mining. Chichester: Horwood Publishing; 2007.
39. Xiangliang Z, Furtlehner C, Germain-Renaud C, Sebag M. Data stream clustering with affinity propagation. Knowl Data Eng IEEE Trans o. 2014;26:1644–56.