Clustering Big Data streams: recent challenges and contributions

1 Introduction

Clustering is a well-established data mining concept that aims at automatically grouping similar data objects while separating dissimilar ones. This process is strongly dependent on the notion of similarity, which is often based on some distance measure. Thus, similar objects are usually close to each other while dissimilar ones are far from each other. The clustering task is performed without a previous knowledge of the data, or in an unsupervised manner.

During the early stages of data mining research, the whole data objects were considered to be statically and permanently stored in the memory. This allowed the designed data mining technique to perform as much passages over the objects as needed to deliver the desired patterns. In the era of big data, the recent growth of the data size and the easiness of collecting data made the previous settings no more convenient. The size of the continuously generated data and the limited storage capacity allow in many scenarios for a single passage over the data, and users are interested in gaining a real time knowledge about the data as they are produced.

A data stream is an ordered sequence of objects that can be read once or very small number of times using limited processing and computing storage possibilities. This sequence of objects can be endless and flows usually at high speeds with a varying underlying distribution of the data. This fast and infinite flow of data objects does not allow the traditional permanent storage of the data and thus multiple passages are not any more possible. Many domains are dealing essentially with data streams. The most prominent examples include network traffic data, telecommunication records, click streams, weather monitoring, stock trading, surveillance data, health data, customer profile data, and sensor data. There is many more to come. A very wide spectrum of real-world streaming applications is expanding. Particularly in sensor data, such applications spread from home scenarios like the smart homes to environmental applications, monitoring tasks in the health sector [14, 20], in the digital humanities using eye-tracking [9] or gesture monitoring [3], but do not end with military applications. Actually, any source of

*Corresponding author: Marwan Hassani, RWTH Aachen University, Data Management and Data Exploration Group, D-52074 Aachen, Germany, e-mail: hassani@cs.rwth-aachen.de

Thomas Seidl: Ludwig-Maximilians-Universität (LMU) Munich, Database Systems Group, D-80538 München, Germany
information can easily be elaborated to produce a continuous flow of the data. Another emerging streaming data sources are social data. In a single minute, 546,000 tweets are happening, 2,460,000 pieces of content are shared on Facebook and 72 h of new videos are uploaded to YouTube. Users are interested of gaining the knowledge out of these information during their same minute of generation. A delay, say till the next minute, might result in an outdated knowledge.

In Figure 1(a), a body sensor network is producing multiple streams about the health status of the runner. Other sensors are collecting streams of other contextual information like the weather and location information. These can be processed on a local mobile device or a remote server to gain, for instance, some knowledge about the near-future status [20]. Figure 1(b) presents another type of sensor streaming data where an eye-tracking system is used to record the duration and the position of each eye fixation over the monitor during a human reading or writing process [21]. One task could be here finding interesting patterns that represent important correlations between eye-gazes and key strokes [9].

Stream clustering aims at detecting clusters that are formed out of the evolving streaming objects. These clusters must be continuously updated as the stream emerges to follow the current distribution of the data. These clusters represent mainly the gained knowledge out of the clustering task. In this article, advanced stream clustering models are introduced. These models are mainly motivated by the basic challenges that we have observed for clustering of streaming data in real world scenarios, particularly sensor streaming data (cf. Figure 1).

The remainder of this article is organized as follows: in Section 2 we will list some of the challenges one has to face while designing stream clustering algorithms for mining big data. In Section 3, we present, at a high level of abstraction, recent approaches in the field of big data stream mining that overcame the challenges mentioned in Section 2. We spend however more time to explain two recent anytime subspace stream clustering in more details. Finally, Section 4 concludes this article with a summarization table of the most important properties of discussed stream clustering algorithms.

2 Challenges of stream clustering of Big Data

Designing stream clustering approaches has some unique special challenges. We list in the following some
paradigms that make it challenging to design a stream clustering approach.

2.1 Adaptation to the stream changes, and outlier awareness

The algorithm must incrementally cluster the stream data points to detect evolving clusters over the time, while forgetting outdated data. New trends of the data must be detected at the same time of their appearance. Nevertheless, the algorithm must be able to distinguish new trends of the stream from outliers. Fulfilling the up-to-date requirement contradicts the outlier awareness one. Thus, meeting this tradeoff is one of the basic challenges of any stream clustering algorithm.

2.2 Storage awareness, and high clustering quality

Due to the huge sizes and high speeds of streaming data, any clustering algorithm must perform as few passages over the objects as possible. In most cases, the application and the storage limitations allow only for a single passage. However, high quality clustering results are requested to make the desired knowledge out of the data stream. Most static clustering models tend to deliver an initial, sometimes random, clustering solution and then optimize it by revisiting the objects to maximize some similarity function. Although such multiple-passages possibility does not exist for streaming algorithms, the requirement of an optimized, high quality clustering does still exist.

2.3 Efficient handling of high-dimensional, different-density streaming objects

The current huge increase of the sizes of data was accompanied with a similar boost in their number of dimensions. This applies of course to streaming data too. For such kinds of data with higher dimensions, distances between the objects grow more and more alike due to an effect termed curse of dimensionality [4]. According to this effect, applying traditional clustering algorithms in the full-space merely will result in considering almost all objects as outliers, as the distances between them grow exponentially with their dimensionality $d$. The latter fact motivated the research in the area of subspace clustering over static data in the last decade, which searches for clusters in all of the $2^d - 1$ subspaces of the data by excluding a subgroup of the dimensions at each step. Apparently this implies higher complexity of the algorithm even for static data, which makes it even more challenging when considering streaming data.

Additionally, as the stream evolves, the number, the density and the shapes of clusters may dramatically change. Thus, assuming a certain number of clusters like in $k$-means-based clustering models or setting a static density threshold as in the DBSCAN-based clustering models is not convenient for a stream clustering approach. A self-adjustment to the different densities of the data is strongly needed while designing a stream clustering algorithm (cf. Figures 2 and 3). Again, this requirement is in conflict with the storage awareness necessity.

2.4 Flexibility to varying time allowances between streaming objects

An additional, natural characteristic of data streams (e.g. sensor data) is the fluctuating speed rate. Streaming data objects arrive usually with different time allowances between them, although the application settings would assume a constant stream speed. Available stream clustering approaches, called budget algorithms in this context, strongly restrict their model size to handle minimal time allowance to be on the safe side (cf. Figure 4). In the case of reduced stream speed, the algorithm remains idle during the rest of the time, till the next streaming object arrives. Anytime mining algorithms, designed recently for static data, try to make use of any given amount time to deliver some result. Longer given times, imply higher clustering quality. This idea was adopted for clustering streaming data. Although this setting can be seen as an opportunity for improving the clustering quality rather than a challenge, it is not trivial to have a flexible algorithmic model that is able to deliver some result even with very fast streams.

3 Recent contributions in the field of efficient clustering of Big Data streams

In this section, we present, at a high level of abstraction, our novel, efficient stream clustering algorithms that consider all of the above challenges mentioned in Section 2. We spend more time to explain two algorithms that contribute to anytime and subspace stream clustering. These contributions [8] are structured in the following
three subsections. In Section 3.1, we present novel high-dimensional density-based stream clustering techniques. In Section 3.2, we introduce and deeply explain advanced anytime stream clustering approaches. Finally, in Section 3.3, we present a unique subspace stream clustering framework as well as the subspace cluster mapping evaluation measure. In all of the following subsections, the first and the second challenges mentioned in Sections 2.1 and 2.2 are carefully considered. Each of the rest of the challenges (Sections 2.3 and 2.4) is the main focus in one of the first two subsections in the following.

3.1 High-dimensional, density-based stream clustering algorithms

In this line of research to address big data stream clustering, we present three density-based stream clustering algorithms. Here, the third challenge mentioned in Section 2.3 is mainly considered.

An efficient projected stream clustering algorithm is introduced for handling high-dimensional, noisy, evolving data streams is presented in PreDeConStream [16]. This technique is based on a two-phase model available in most stream clustering algorithms [1, 6] (cf. Figure 3). The first phase represents the process of the online maintenance of data summaries, called microclusters. As the name suggests, those summaries are smaller clusters, where only minimal statistics of the streaming objects are collected. These statistics are the count, the linear sum and the squared sum. The microclusters are then passed to an offline phase for generating the final clustering. The technique works on incrementally updating the output of the online phase stored in a microcluster structure. Taking those microclusters that are fading out over time into consideration speeds up the process of assigning new data points to existing clusters. The algorithm localizes the change to the previous clustering result, and smartly uses a clustering validity interval to make an efficient offline phase.

HASTREAM [17] contributes a hierarchical, self-adaptive, density-based stream clustering model (cf. Figure 2). The algorithm focuses on smoothly detecting the varying number, densities and shapes of the streaming clusters. A cluster stability measure is applied over the summaries of the streaming data (the microclusters in Figure 3), to extract the most stable offline clustering.

In I-HASTREAM [15], in order to improve the efficiency of the suggested model in the offline phase, some methods from the graph theory are adopted and others were contributed, to incrementally update a minimal spanning tree of microclusters (cf. the red arrows in Figure 2). This tree is used to continuously extract the final clustering, by localizing the changes that appeared in the stream, and maintaining the affected parts merely.

3.2 Advanced anytime stream clustering algorithms

By considering all other challenges, the main focus of the two algorithms presented in this section are the third and
the fourth challenges mentioned in Sections 2.3 and 2.4. Anytime algorithms build upon the realistic assumption of the varying time allowances between streaming objects (cf. Figure 4). They aim at increasing the quality of their output if they were given more time (i.e. the time allowance $\Delta t$ is bigger) instead of being idle as in traditional algorithms.

The LiarTree algorithm [13] is contributed on the online phase (cf. Figure 3) to provide precise stream summaries and to effectively handle noise, drift and novelty at any given time. The algorithms stores the microclusters in a tree data structure (cf. Figure 5). The microclusters are indexed in a hierarchical way, such that microclusters in any higher level of the tree are bigger and less in number than the ones in lower levels. The most fine-grained microclusters are stored in the leaf level of the tree. Additionally, each microcluster points to the smaller microclusters that are nested inside it. Thus, the insertion of objects inside its microcluster in the leaf-level is performed in an efficient way. We prove that the runtime of the our anytime algorithm is logarithmic in the size of the maintained model opposed to a linear time complexity often observed in previous approaches. The anytime concept is achieved using the insertion of objects in the tree. If $\Delta t_i$ (cf. Figure 4) is big enough, the object $o_i$ can be inserted into its leaf level microcluster, as this insertion needs more time but guarantees a better quality. If $\Delta t_i$ is however small (i.e. another object $o_{i+1}$ has arrived in the stream), then $o_i$ is stored temporarily in a buffer at the level where it has reached, and the insertion of $o_{i+1}$ is prioritized. The buffered $o_i$ is “hitchhiked” later once there is another insertion of a later object $o_j; j > i$ descending in its same insertion path to its destination.

Three other main contributions of LiarTree, the first is enabling the anytime concept to fast adapt to the new trends of the data by allowing a liar growth of the tree temporarily in a bottom-up way to allow faster splitting, merging and birth of new microclusters. The second is filtering noise from new trends of the stream by allowing a local buffering of noise objects until they their density is enough (w.r.t. neighboring microclusters) to form a new microcluster. The final contribution is guaranteeing a logarithmic complexity of the insertion.

In the SubClusTree algorithm [12], another complexity dimension is added to the problem addressed in LiarTree [13]. The high-dimensionality paradigm of big streaming data (cf. Section 2.3) is considered together with the varying arrival times and the streaming aspects of the data (cf. Section 2.4). SubClusTree is a subspace anytime stream clustering algorithm, that can flexibly adapt to the different stream speeds and makes the best use of available time to provide a high quality subspace clustering. It uses a forest of multiple liartrees as its data structure (cf. Figure 6). Each liartree represents here a subspace. Thus, in the illustrative example in Figure 6 where a 3D dataset is considered, 110, 010 and 001 represent the liartrees where the insertion is performed over the subspace that contains the first dimension, the second and the third, respectively.

As explained using the blue arrows in Figure 6, if $\Delta t_i$ is big enough, the insertion of $o_i$ proceeds to the subspaces (liartrees) of two dimensions, and so on, until reaching the tree of the full space.

The insertion within a single liartree is not allowed to be interrupted until the object reaches its leaf level. Once $\Delta t_i$ has finished, the inserted dimensions of $o_i$ are only considered, and the insertion of $o_{i+1}$ is prioritized. Apparently it is not convenient for SubClusTree to have a data
structure of $2^d - 1$ liartrees for processing a $d$-dimensional dataset, thus liartrees of only populated subspaces are initiated. Usually those relevant trees are considerably smaller than the full number of trees. Popular subspaces are decided over a certain batch using a heuristic that decides potential higher-dimensionality subspaces as the combination of popular lower-dimensional subspaces in an Apriori-like method.

The heuristic used by SubClusTree estimates the density of flexible grids to efficiently distinguish the populated higher-dimensional subspaces (i.e., subspaces with clusters) from irrelevant ones.

3.3 A framework and an evaluation measure for subspace stream clustering

After the development of the previous algorithms, a new gap in the literature of evaluating stream subspace clustering algorithms has appeared.

The first subspace clustering evaluation framework over data streams, called Subspace MOA, is presented in [11]. This open-source framework is based on the MOA stream mining framework [5], and has three phases (cf. Figure 7). In the online phase, users have the possibility to select one of three most famous summarization techniques to form the microclusters. Upon a user request for a final clustering, the regeneration phase constructs the data objects out of the current microclusters. Then, in the offline phase, one of five subspace clustering algorithms can be selected. In addition to the previous combinations, the framework contains available projected stream clustering algorithms like PreDeConStream [16] and HDDStream [19]. The framework is supported with a subspace stream generator, a visualization interface and various subspace clustering evaluation measures.

With the increase of the size of high-dimensional data, applying traditional subspace clustering algorithms is impossible due to their exponential complexities. Figure 8(a) shows, for instance, that applying the static variant of PROCLUS [2] over a small subset of the KDD dataset [7] is extremely slow even on a reasonable machine. Beginning from a size of 200 K objects only, the experiment does not finish. However, as shown in Figure 8(b), it is possible by using Subspace MOA over the same machine to get the relatively-large dataset completely clustered with the same subspace clustering algorithm (PROCLUS [2]).

In [10], a novel external evaluation measure for stream subspace clustering algorithms called SubCMM: Subspace Cluster Mapping Measure, is presented. SubCMM is able to handle errors caused by emerging, moving, or splitting subspace clusters. This first evaluation measure that is designed to reflect the quality of stream subspace algorithms is directly integrated in the Subspace MOA framework. The experimental evaluation, performed using the Subspace MOA framework, depicts the ability of SubCMM to reflect the different changes happening in the subspaces of the evolving stream.
Table 1: Properties of the different stream clustering algorithms discussed in this article (\(na = \text{not applicable}\)).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Online-offline model</th>
<th>Density-based (in online phase)</th>
<th>Density-based (in offline phase)</th>
<th>Hierarchical</th>
<th>Outlier-aware (cf. Section 2.1)</th>
<th>Density-adaptive (cf. Section 2.3)</th>
<th>Anytime (cf. Section 2.4)</th>
<th>Incremental (in offline phase)</th>
<th>Overlapping clusters and subspaces</th>
<th>Data structure of microclusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-space stream clustering algorithms</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Pyramidal time frame</td>
</tr>
<tr>
<td>CluStream [1]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DenStream [6]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LiarTree [13]</td>
<td>✓</td>
<td>✓</td>
<td>na</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>List</td>
</tr>
<tr>
<td>HASTREAM [17]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>na</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I-HASTREAM [15]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>na</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>na</td>
<td></td>
<td></td>
<td></td>
<td>List or Index</td>
</tr>
<tr>
<td>Subspace stream clustering algorithms</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PreDeConStream [16]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>List</td>
</tr>
<tr>
<td>HDDStream [19]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SubClusTree [12]</td>
<td>✓</td>
<td>✓</td>
<td>na</td>
<td>✓</td>
<td></td>
<td>na</td>
<td></td>
<td></td>
<td></td>
<td>Index</td>
</tr>
</tbody>
</table>

4 Conclusion

In this article, we have mainly addressed the following three v’s of big data: velocity, volume and variety. Various streaming data applications, with a huge volumes and varying velocities were considered in different scenarios and applications.

A list of the recent challenges that face the designer of a stream clustering algorithm was shown and deeply discussed. Finally, recent contributions on three main research lines in the area of big data stream mining, were presented and their fulfillment to the design requirements was highlighted.

Table 1 summarizes the properties of the stream clustering algorithms discussed in this article w.r.t. different aspects. It checks whether the specific algorithm follows the online-offline model introduced in Figure 3. Also, the algorithms that apply a density-based clustering concept on their online or offline phases are listed. Additionally, the algorithms delivering a hierarchical final clustering are highlighted. All algorithms are then checked whether they fulfill the challenges introduced in Sections 2.1, 2.3 and 2.4. For the subspace stream clustering algorithms, the table checks additionally, whether they produce simultaneously overlapping clusters and subspaces [18]. Projected stream clustering algorithms like HDDStream [19] and PreDeConStream [16] produce non-redundant clusters, where any object from the dataset can be maximally a part of one projected cluster. Finally, the data structure used by each algorithm for storing and accessing the microclusters is mentioned. CluStream [1] uses a pyramidal time frame data structure to store snapshots of the microclusters at carefully selected timestamps. The other algorithms efficiently use a list structure, a tree or both to access their microclusters.

References


Bionotes

Dr. Marwan Hassani
RWTH Aachen University, Data Management and Data Exploration Group, D-52074 Aachen, Germany
hassani@cs.rwth-aachen.de

Dr. Marwan Hassani is postdoc researcher and associate teaching assistant at the data management and data exploration group at the RWTH Aachen University. His research interests include stream data mining, sequential pattern mining of multiple streams, efficient anytime clustering of big data streams, streaming process mining and exploration of evolving graph data. Marwan received his PhD (2015) from RWTH Aachen University. He received an equivalence Master in Computer Science from RWTH Aachen University (2009). He coauthored more than 40 scientific publications and serves on several program committees.

Prof. Dr. Thomas Seidl
Ludwig-Maximilians-Universität (LMU) Munich, Database Systems Group, D-80538 München, Germany
seidl@dbs.ifi.lmu.de

Prof. Dr. Thomas Seidl is full professor for computer science and head of the database systems group at Ludwig-Maximilians-Universität (LMU) Munich. His research activities include data mining and database technology for multimedia and spatial-temporal objects in engineering, business, life science and humanities applications. He coauthored more than 300 scientific publications and serves on several program committees and scientific boards. Seidl received his Diplom (MSc, 1992) from TU München and his PhD (1997) and venia legendi (2001) from LMU Munich. From 2002 to 2016 he headed the data management and data exploration group as a full professor for computer science at RWTH Aachen University.