Efficient Streaming Detection of Hidden Clusters in Big Data Using Subspace Stream Clustering

Marwan Hassani(✉) and Thomas Seidl

Data Management and Data Exploration Group, RWTH Aachen University, Aachen, Germany
{hassani,seidl}@cs.rwth-aachen.de

Abstract. Recently, many data mining techniques were revisited to cope with the new big data challenges. Nearly all of these algorithms considered the efficiency of the mining algorithm to survive the increasing size of the data. However, as the dimensionality of the data increases, not only the efficiency but also the effectiveness of traditional mining algorithms is compromised. For instance, clusters hidden in some subspaces are hard to be detected using traditional clustering algorithms, as the dimensionality of the data increases. In this paper, we consider both the huge size, and the high dimensionality of big data by providing a novel solution that presents a three-phase model for subspace stream clustering algorithms. Our novel model, overcomes the huge size of the big data in its first phase, by continuously applying a streaming concept over the huge data objects, and summarizing them into micro-clusters. Then, after each certain batch of data, or after upon a user request, the second phase is applied over the data summarized in micro-clusters, to reconstruct the current distribution of the data out of the current summaries. In the third phase, a subspace clustering algorithm is applied to overcome the high dimensionality of the data, and to find hidden clusters within some subspace. An extensive evaluation study over different scenarios that follow our model over a big data set is performed.

Keywords: Subspace stream clustering · Streaming big data · Real-time streaming analysis of big data

1 Introduction

The continuously-increasing sizes and dimensionality of available data is a fact that is appearing in a lot of domains nowadays. Actually, it is very hard to find a data producing area where the output does not easily become big, with respect to the size, the dimensionality, or both. Recently, many data mining techniques were revisited to cope with the emerging big data challenges. Almost all of these algorithms considered the efficiency of the mining algorithm to survive the increasing size of the data. However, as the dimensionality of the data increases,

W.-S. Han et al. (Eds.): DASFAA 2014, LNCS 8505, pp. 146–160, 2014.
DOI: 10.1007/978-3-662-43984-5_11,

not only the efficiency, but also the effectiveness of traditional mining algorithms is compromised. In this paper, we present an algorithmic model that considers both the big size and the high dimensionality of the data using the subspace stream clustering concept. We shortly define the related concepts in Sects. 1.1 and 1.2, then we present the motivation with a paper structure in Sect. 1.3.

1.1 Subspace Clustering

Clustering is an example of a data mining algorithm that aim at grouping similar objects in the data set in the same group or cluster, and dissimilar objects in different groups or clusters. The definition of *similarity* here is introduced according to some distance function. Thus, far objects are dissimilar, and vice versa. However, for big data, as the dimensionality of the objects increases, the distances between objects in higher subspaces increases exponentially, making it very hard to find "similar" objects, and all objects of the data set appear as outliers. This problem was termed "curse of dimensionality" [5] where traditional distance functions are no more applicable.

In recent applications like network intrusion detection, objects (connections) are described using many dimensions. For example, each connection in the famous Network Intrusion Dataset [1] has 42 dimensions. Recent research introduced subspace clustering which aims at locally detecting relevant dimensions per cluster. If objects in a certain cluster are densely close to each other on some subgroup of dimension (also called *subspace*), then this subgroup of dimensions is relevant to that cluster. Otherwise, if they are scattered over other dimensions then those dimensions are irrelevant to that cluster. Thus, for each cluster, relevant dimensions are locally determined and irrelevant dimensions are ignored.

Figure 1 gives an example of applying a full-space clustering of objects with only two dimensions, for illustrative purposes. *Cluster 1* represents the only full space cluster (here 2-*dimensional* cluster) as both dimensions are relevant. All other points are considered outliers, since they are not forming a dense area in the full space. However, as we see in Fig. 2, for the same dataset, although the purple objects are categorized as outliers by full-space clustering in Fig. 1, they form a dense cluster when projecting them over *Dim 2*. Objects of *Cluster 2* are highly scattered in *Dim 1* making *Dim 1* irrelevant for *Cluster 2*. Subspace/Projected clustering algorithms try to find all clusters hidden in any subspace, and not only in the full space, in addition, to the full space clusters.

The same discussion applies to the projections of *Cluster 1* over *Dim 1* and *Dim 2*.

1.2 Stream Mining

Streaming data on the other hand, is available in increasingly many applications. In modern IT-systems, there is a constant stream of data sprouting out of log and maintenance files. The number of attributes in these files still rises, together with the frequency of output generation. The same applies to connection protocols

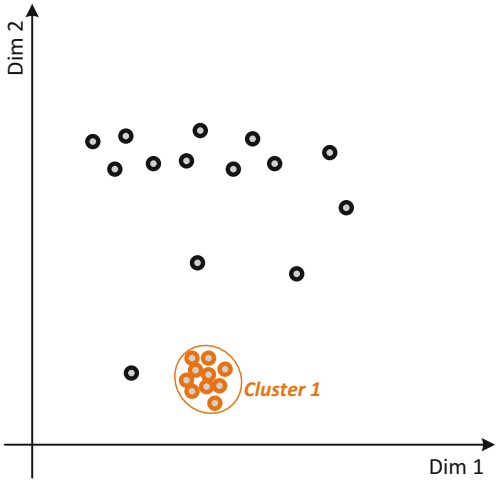

Fig. 1. An example of full-space clustering

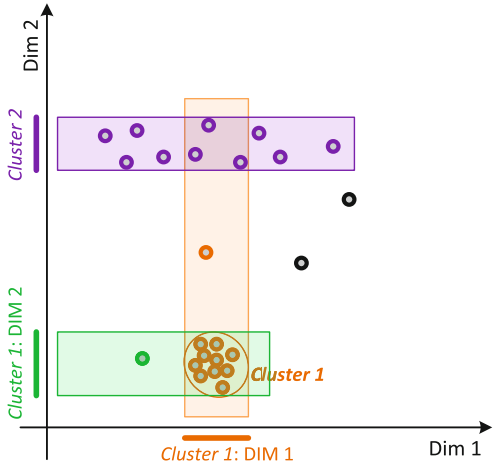

Fig. 2. An example of subspace clustering

of different user groups, be it phone-calls or TCP-connections, as well as monitoring user behavior of a software in a live environment. There are thousands of examples about sensors measuring different sorts of attributes like pollution, micro waves, humidity, heart beats, body temperature, acceleration, speed, location etc. In all of the above examples, there is constant stream of information which has to be processed in a timely manner. Apparently, it is impossible to save every bit of information getting caught in major telescopes, the same goes for large-scale physical experiments where saving all incoming data might not be possible or would delay the next experiment by a large margin.

All these applications have one thing in common: To further get a grasp of the information contained deep in the data, we have to handle a constant flood of data. Sometimes it is not applicable to save the data stream due to memory restrictions or it might be of massive importance to mine information as soon as they occur. In all these cases, we somehow have to work with streaming, volatile data and use procedures to maximize the information gain we can get along while paying attention to occupy just enough space to make it happen.

1.3 Motivation

The streaming model of mining data online was a suitable concept to be adopted when mining big *static* data. The idea of online processing of the data with the expectation of different speeds is very suitable for processing data over batches. Although, such solution might reduce the quality of the mining algorithm due to its possibility of real time processing and management of the data, the ability to handle huge data sizes batch-wise made them appealing for the big data community.

To handle the challenge of increasing dimensionality in big data, we will consider also the usage of subspace clustering to find hidden clusters of data within some subspace. Available subspace clustering algorithms (cf. Sect. 2) are known to be heavy-weighted when talking about scanning the whole data. This feature contradicts the stream mining requirements of having a light-weighted version of any algorithm, that is capable of processing the data online.

In this paper, we will consider both the huge size, and the high dimensionality of the big data by providing a novel solution that presents a model for subspace stream clustering algorithms. Our novel model, overcomes the huge size of the big data by continuously applying a streaming concept over the huge data, and summarizing them into micro-clusters. Then, after each certain batch of data, or after a user request, the data is reconstructed out of the micro-clusters summaries and forwarded to one subspace clustering algorithm, to overcome the high dimensionality of the data, and to be able to find hidden clusters within some subspace.

The remainder of this paper is organized as follows: Sect. 2 lists some of the related work in the areas of stream clustering and static subspace clustering. Section 3 introduces our model and brings a running example to explain it step-by-step. Section 4 presents a thorough evaluation that compares the running time and the clustering quality of different scenarios of our given model and compares them to one static model. Then we conclude this paper in Sect. 5 by giving also an outlook.

2 Related Work

The first grid based subspace clustering algorithm introduced was CLIQUE [4]. It works with a grid-based approach to identify subspace clusters in a bottom-up fashion using so called dense grid cells which contain more points than a

certain threshold. This is done with using the apriori method. To speed up the computation, monotonicity laws are used to prune the possible dense-regions. This idea is picked up by MAFIA [10] but instead of using a static grid, the grid is now adaptive and pruning is dropped in favor of heavy parallel optimization. Projected clustering uses a special distance function in conjunction with a common clustering algorithm. PROCLUS [3] is a three-phase algorithm, and this is the one we use in this paper as a running example. First we guess medoid candidates from objects spread over the data set. Then we improve these medoids and compute dimensions for each. In the last phase, we use an algorithm similar to k-medoid to refine the clusters. *P3C* [15] starts with one-dimensional intervals which might be approximate higher dimensional subspace clusters. Merging these in an a-priori bottom-up method nets high-dimensional subspace clusters. These maximal-dimensional clusters are used as cluster-cores for a refinement step using expectation-maximization-like algorithm. SubClu [14] uses the DBSCAN model [9] of connected sets in an apriori style. By applying DBSCAN over each subspace, SubClu requires a high runtime.

Our algorithmic model presented in this paper differs from the above algorithms by applying the subspace clustering algorithm over streaming data. Due to their huge complexity, a straight-forward utilization of the above algorithms over big datasets as in the streaming cases faces serious memory issues (cf. Sect. 4). Our algorithmic model gives a smooth way of applying subspace clustering algorithms in the streaming scenario.

There is a rich body of literature on stream clustering. Approaches can be categorized from different perspectives, e.g. whether convex or arbitrary shapes are found, whether data is processed in chunks or one at a time, or whether it is a single algorithm or it uses an online component to maintain data summaries and an offline component for the final clustering (as most of stream clustering algorithms). Convex stream clustering approaches are based on a k-center clustering [2,13]. Detecting clusters of arbitrary shapes in streaming data has been proposed using density based clustering [7,8]. Another line of research considers the anytime clustering with the existence of outliers [12].

The model we present in this paper follows a three-phase model that differs from the two-phase one which appears in most of the above stream clustering algorithms. This is done to be able to deal with big data requirements as well as the subspace clustering algorithms requirements.

3 Algorithmic Model

In this section we will present the model we are using for applying subspace stream clustering algorithms to efficiently and effectively find subspace clusters in big data. We will use one scenario of the combination: CluStream+SUBCLU as a running example, while the same will apply for other scenarios. In the experimental part, we will show a comparison between most scenarios in details.

The idea of stream subspace clustering is intuitive: We cluster the incoming data "live" and save features of the clustering for a defined period of time. We can

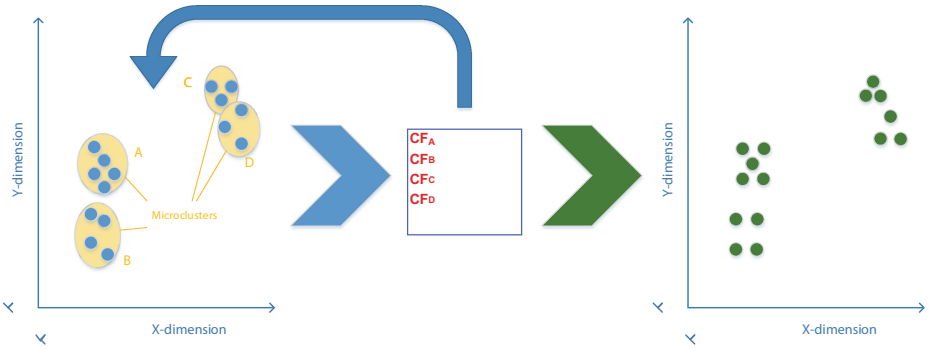

Fig. 3. The whole process of our model for stream subspace clustering for big data. The blue arrows represent the online phase while the green arrow represents the reconstruction and the offline phases (Color figure online).

then approximately reconstruct the data for a given time frame, the more recent the more accurate, and use a classic subspace algorithm to determine a clustering of this frame.

Going a bit more into detail, we use a three-stop approach in this paper: an online phase, a regeneration phase and then an offline phase. The data stream gets processed by CluStream [2] (or DenStream [7]) in the online phase, which produces micro clusters for the current input data (cf. Sect. 3.1). These clusters are then saved as cluster feature vectors as seen in Fig. 3. Then, upon some request from the user for a final clustering or after a certain amount of time, we regenerate the points out of the summaries in the regeneration phase (cf. Sect. 3.2). The regenerated data is then forwarded to the final subspace clustering algorithm which produces the final clusters (cf. Sect. 3.3).

3.1 Online Phase: A Stream Clustering Algorithm

In the online phase of our model, a summarization of the data stream points is performed and the resulting microclusters is given by sets of cluster features:

$$CF_A = (N, LS_i, SS_i)$$

which represent the number of points within that microcluster A, their linear sum and their squared sum, respectively. One of the two online algorithms (CluStream or DenStream)is responsible for forming these microclusters, deleting older ones or continuously maintaining the updated ones.

3.2 Regeneration Phase: Gaussian Out of Online Summaries

After reaching a predefined time threshold, we call it here (window size), we compute Gaussian distributed objects out of the statistics we got from the cluster features of the microclusters (green arrow Fig. 3).

This step is called the offline phase, where the clustering features are used to reconstruct an approximation to the original N points, for each microcluster, using Gaussian functions to reconstruct points over each dimension i.

$$c_i = \frac{LS_i}{N}$$

with a radius:

$$r = \sqrt{\frac{SS}{N} - (\frac{LS}{N})^2}$$

where:

$$SS = \frac{1}{d}\sum_{i=1}^{d} SS_i$$

and

$$LS = \frac{1}{d}\sum_{i=1}^{d} LS_i$$

The generated N_A points for each microcluster will be now normally distributed. Thus, they will look a little bit differently distributed than the original distribution (compare the points in Fig. 3 Right with Left). Actually, this is the only approximation that we have in our model.

3.3 Offline Phase: A Subspace Clustering Algorithm

The generated N points are forwarded to one of the four subspace clustering algorithms. These are SubClu [14], ProClus [3], Clique [4] and P3C [15]. This results with 8 different combinations of algorithms that can be tested. These algorithms are applied to the streaming cases. Other than static data that do not vary over time, stream data are given in different rate and pattern changing dynamically, which makes it challenging to analyze its evolving structure and behavior. In streaming scenarios, we also often face limitations on processing time and storage, since a vast amount of continuous data are coming rapidly.

The offline part of our model uses a subspace clustering algorithm to deliver the final clusters. In our running example we use the subspace clustering algorithm: SUBCLU [14] over the regenerated points. The aim of SUBCLU is to find all clusters in all subspaces of the data in a greedy, bottom-up way. The rough work-flow of the algorithm is that we start in the 1-dimensional subspaces and find clusters using DBSCAN. Then we recursively compute a set of candidate subspaces with the dimensionality $k + 1$, prune these and test each candidate if there are still clusters left in the subspace. As there are a lot of range queries necessary for DBSCAN which make up the bulk of computation time needed in SUBCLU, the implementation introduced in [14] uses an efficient index support for range queries of single attributes in logarithmic time. In case of range queries on more than one attribute, the range query for each attribute is used on its own and the intersection of all intermediate results is computed as final result.

4 Experiments

4.1 Example

We test this combination of CluStream and SUBCLU on a very large dataset containing connection information. Each of these objects contains 41 different attributes from which we will try to cluster information. 4.8 million objects in total are included in this data set, To compare SUBCLU and other offline-algorithms, we will test a variety of other macro algorithms in regard of accuracy and performance. This will be done using the Subspace MOA framework [11], as it gives the possibility to work with a GUI and all common offline-algorithms, such as CLIQUE, PROCLU and P3C. All calculations were done on a AMD FX 8-core clocked at 4 GHz with 8 GB RAM.

4.2 Dataset

The real data set used in this experiment is the KDD CUP'99 Network Intrusion Detection data set [1] which has been used to evaluate several stream clustering algorithms [2] and [7]. The MIT Lincoln Labs recorded the traffic of a LAN network for two weeks. Each connection is labeled as either normal, or as an attack. There are 22 different types of attacks, which fall into the following four main categories:

1. DOS: denial-of-service
2. R2L: unauthorized access to a remote machine
3. U2R: unauthorized access to local root privileges
4. Probing: surveillance and other probing.

The data set consists of 494021 TCP connections where most of them are normal connections. Each connection has 42 attributes. These attributes can be discrete or continuous. As in [2] and [7], all the 34 continuous attributes for the clustering task are used. Table 1 gives a detailed view of the different attacks appearing within a horizon $H = 5$ and stream speed = 1000 of this dataset.

4.3 Framework

The software used as part of this paper is the java based Subspace MOA. Introduced in [11], this software uses the interface of the MOA framework [6] style and contains an additional tab which is solely for subspace clustering. As data input Subspace MOA supports a synthetic random RBF (subspace) generator or the option to read an external ARFF file as input stream. As for one-stop algorithms, seven total different algorithms can be chosen, for the three-phase method, three different online- as well as five common offline algorithms are available.

Table 1. Table of stream of Network Intrusion Data set within the horizon $H = 5$, stream speed = 1000

Normal or attack type	Objects within horizon $H = 5$ at time unit			
	150	350	373	400
Normal	4004	4097	892	406
Satan	380	0	0	0
Buffer overflow	7	1	2	0
Teardrop	99	99	383	0
Smurf	143	0	819	2988
Ipsweep	52	182	0	0
Loadmodule	6	0	0	1
Rootkit	1	0	0	1
Warezclient	307	0	0	0
Multihop	0	0	0	0
Neptune	0	618	2688	1603
Pod	0	1	99	0
Portsweep	0	1	117	1
Land	0	1	0	0
Sum	5000	5000	5000	5000

CluStream is based on snapshots, saved as extended cluster feature vectors which contain information about characteristics of the discovered micro clusters. Contained information be contained are the sum of data values and time stamps amongst others. The total amount of micro clusters is always maintained, as such new objects are inserted in the closest already existing micro cluster (MC) or a new micro cluster containing this object is created. As the amount of micro clusters has to be constant, a not-relevant micro cluster is then deleted or the two closest cluster are merged. Using a pyramidal time frame to store snapshots allows us to halt the computation at any given point and produce the input for SUBCLU.

For this part, we have to specify the time-horizon h. For this horizon, CluStream produces points for each micro cluster which are computed according to the variance and mean of the saved micro cluster information using a Gaussian distribution. This set of points is then passed to SUBCLU.

SUBCLU then produces the actual clustering using the input provided by CluStream. Since only the snapshot of the recent data is passed over, we will most likely not find all possible subset clusters in any given snapshot.

In the following, we will take a closer look what the used implementation of CluStream in conjunction with SUBCLU in the SubspaceMOA framework can accomplish. In the first section we will compare SUBCLU against CLIQUE, PROCLUS and P3C and get a first glimpse of accuracy between those. In the following section, we will compare different settings of SUBCLU in regards to three measurement methods.

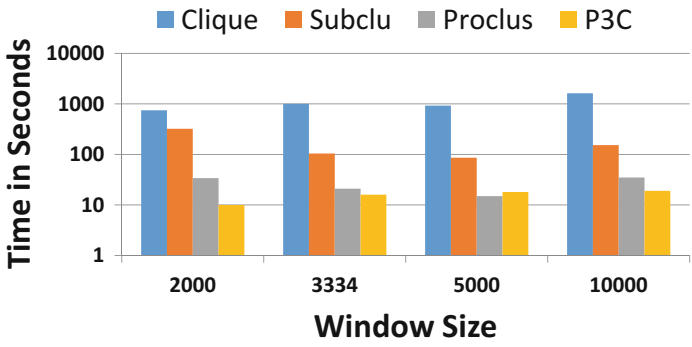

Fig. 4. The performance of the four offline algorithms after using CluStream in the online phase when changing the window size (the batch size).

4.4 Running Time Results

To evaluate the performance we will have a set micro algorithm, CluStream with a maximum of 30 micro clusters, and compare four different window sizes of 2000–10000 for four different macro clustering algorithms while keeping the number of overall processed objects at a steady 10000. CLIQUE with default settings in SubspaceMOA which are $\xi = 10$ and $\tau = 0.01$, SUBCLU with $\epsilon = 0.002$ and $m = 5$, PROCLUS with $c = 5, d = 2$ and P3C with $p = 10, \chi^2 = 0.001$ which also are default settings.

Note that in Fig. 4 the scale of the runtime in s is logarithmic. CLIQUE is too slow to have all values in one figure if using metric scale.

SUBCLUs predecessor CLIQUE is in every aspect the slowest algorithm going over this benchmark. At even the best case, it is slower than any other algorithm in this set. It also needs even more time to process all 10000 objects when using a larger window size peaking at a stunning 27 min needed for one run when using the whole 10000 objects as one window.

SUBCLU is vastly more efficient. Between 80 s and just over 5 min are the values for the different window sizes. Interestingly, a window size of 2000, and thus doing 5 computations of each, yields the worst results while two computation circles of 5000 objects each are still good runtime-wise.

PROCLUS and P3C are both extremely fast when compared to SUBCLU, peaking at 35 s tops for a worst-case window. PROCLUS shows the same preference for a 5000 object window as SUBCLU does, P3C prefers smaller windows.

To observe the huge improvement our model brings when compared to the static subspace clustering algorithms, we tried to apply the KDD CUP '99 dataset over the static PROCLUS algorithm. We have tried first to run the PROCLUS over the whole dataset size with 1 G memory allocated for the algorithms' heap. As it was crashing, we decided to try smaller versions of the dataset, by getting the first ones as they appear in the dataset. As shown in Fig. 5, the exponential increase of the runtime is obvious as the size of the dataset increases. The algorithms started to crash when trying a sub-datset of size 200 K. Additionally,

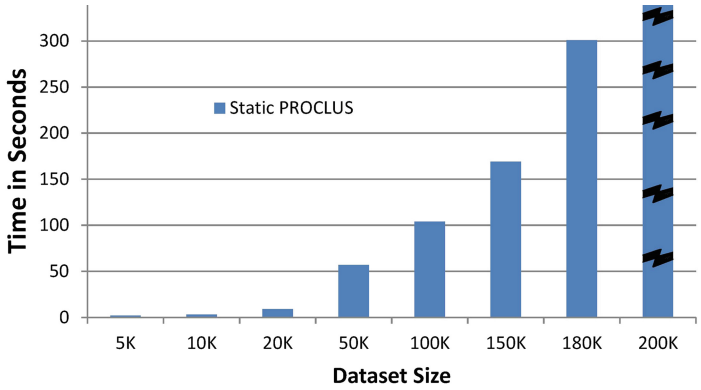

Fig. 5. The runtime performance of a *static* subspace clustering algorithm: PROCLUS. Beginning from a sub-dataset size of 200 K objects, the algorithm failed to successfully finish the running.

the runtime improvement that our algorithmic model causes over PROCLUS is obvious when trying any window size (cf. Fig. 4).

4.5 Accuracy Results

A total of three different evaluation measures are used in this section: The $F1$ measure, which gives a overview how well hidden clusters are represented by the output of the algorithm, RNIA [16], which measures how well hidden subobjects are covered by already found objects, and CE [16], which works similar to RNIA but also evaluates if a cluster is split up into several smaller clusters. From now on, when talking about RNIA or CE measures, we mean $1 - CE$ and $1 - RNIA$, so the nearer the measure to 1, the better, just as it is the case for the $F1$ measure. For the previous settings of the performance evaluation, we averaged for each algorithm the three accuracy evaluation measures. Figure 6 depicts the gained results as the batch size (window size) changes. As expected, the accuracy of almost all algorithms increases when the window size increases, with one exception with SubClu which is fluctuating a bit. The reason of this improvement of the accuracy is the fact that considering more data at a time, gives each algorithm more possibility to find even more hidden clusters. Another observation, is that nearly all of the algorithms who performed well w.r.t. the running time, are also accurate. While the slow ones are also delivering bad results. This makes P3C a winner algorithm when considering the running time and accuracy.

Going over the different settings of CluSream and SUBCLU, we first check for different ϵ to get a feel of how this parameter affects the performance. Just as in the performance section, we use 30 as maximum number of micro clusters for CluStream, 2000 as window size and $m = 5$ for SUBCLU.

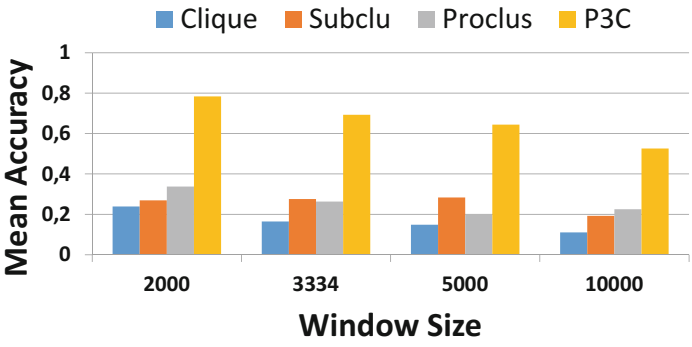

Fig. 6. The averaged accuracy of the four offline algorithms when using CluStream in the online phase when changing the window size (the batch size).

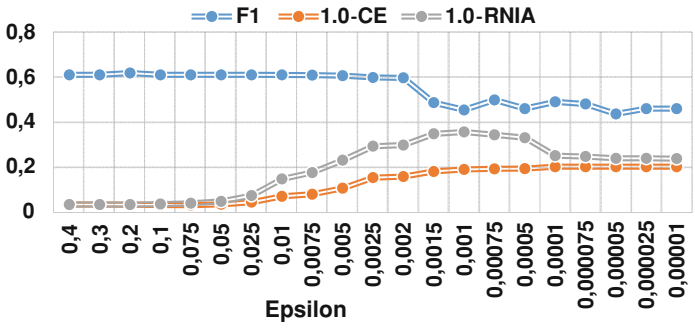

Fig. 7. Different accuracy measures of the SUBCLU algorithm after using CluStream in the online phase when changing ϵ (the neighborhood parameter).

Presented in Fig. 7, the $F1$ measures start to drop after $\epsilon = 0.002$. Prior to this value, smaller settings meant that there are more but smaller clusters holding the balance between precision and recall. After this, a bit of precision is lost, resulting in the 0.1 worse measure. Starting from a smaller $\epsilon = 0.0015$, both RNIA and CE have a maximum with RNIA falling a bit after $\epsilon = 0.0001$. It seems like the algorithm did not find too many objects after this part, however the found ones are clustered with little excessive clusters. Another parameter to check is the minimum points m, found in Fig. 8. For this benchmark, $\epsilon = 0.001$ was used. $m = 9$ seems to be an interesting point, resulting in a spike from both $F1$ and RNIA in opposite directions. We could assume this was a threshold for adding "bad" objects to a cluster without enabling DBSCAN to connect the cluster to an existing "good" one.

Overall, this setting seems to have lower impact on the accuracy of SUBCLU than the ϵ parameter. As these settings are mainly used for DBSCAN this is not unsurprising but still nice to confirm.

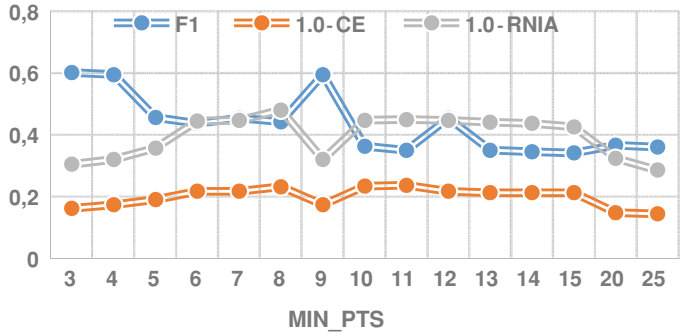

Fig. 8. Different accuracy measures of the SUBCLU algorithm after using CluStream in the online phase when changing m (the minimum number of points needed in the ϵ-neighborhood for an object to become core [14]).

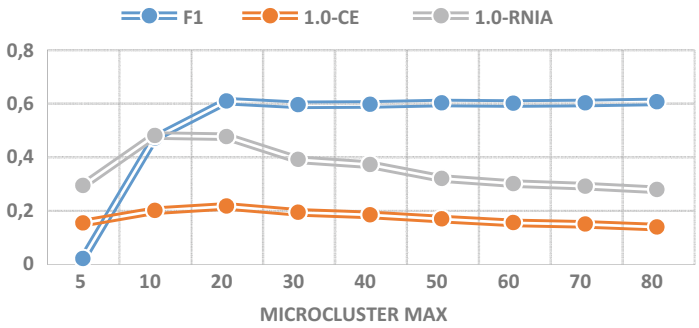

Fig. 9. Different accuracy measures of the SUBCLU algorithm after using CluStream in the online phase when changing the maximum allowed number of microclusters within CluStream.

In this last part of this accuracy evaluation we will take a short look to see how varying different the maximum amount of micro clusters within CluStream affects the quality of the results. Figure 9 depicts the results, as expected, a certain minimum amount of micro clusters has to be present to achieve good results. After this threshold, which seems to be around 20, the results do not change too much but steadily go worse. This holds true especially for the RNIA measure. Probably 23 is a good number because there are a total of 23 different connection types present in the data set.

5 Conclusion and Outlook

In this paper, we considered both the huge size, and the high dimensionality of big data by providing a novel solution that presents a three-phase model for subspace stream clustering algorithms. Our novel model, proved to overcome the

huge size of the big data in its first phase, by continuously applying a streaming concept over the huge data objects, and summarizing them into micro-clusters. Then, after each certain batch of data, the second phase is applied over the data summarized in micro-clusters, to reconstruct them by assuming a Gaussian distribution within the relatively small sized micro-clusters. Using a subspace clustering algorithm in the third phase, to overcome the high dimensionality of the data, we make the heavy subspace clustering feasible in finding hidden clusters within some subspace of big datasets.

In more details, we took a look at the density-based subspace clustering algorithm SUBCLU as a running example. We used important observations about monotonicity to develop the idea of a bottom-up greedy algorithm which could detect arbitrary shaped clusters hidden in subspaces. We then used this algorithm to process a high-dimensional data stream of connection information, using CluStream as micro algorithm in conjunction with SUBCLU as macro algorithm. Comparing SUBCLU to CLIQUE, PROCLUS and P3C, we observed the runtime requirements depending on different window sizes of the micro clustering. After that, we observed how different parameter settings played a large role in producing good results and finished in evaluating the current implementation in Subspace MOA. We have also shown that a direct application of huge datasets over static subspace clustering algorithms will face serious memory issues. Currently, an algorithm such as $P3C$ is fast and accurate. The only problem is inherited from the density-based subspace clustering: the dependency on input parameters done by the user. This lead to a trial and error-style preparation before achieving good results.

One good future direction could be discussing two main ideas. First: how to minimize the effect of the input parameters by optimizing the selection of the density parameters such that they fit the distribution in the current batch of the data. Second: how to minimize the effect of the ordering of the data, such that we can read the big data in any streaming order without a dramatic change of the overall accuracy or performance.

Acknowledgments. This work has been supported by the UMIC Research Centre, RWTH Aachen University, Germany.

References

1. KDD Cup 1999 Dataset. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Accessed 22 Nov 2013
2. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: VLDB '03, pp. 81–92 (2013)
3. Aggarwal, C.C., Wolf, J.L., Yu, P.S., Procopiuc, C., Park, J.S.: Fast algorithms for projected clustering. In: ACM SIGMOD Record, vol. 28, pp. 61–72 (1999)
4. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications. In: SIGMOD '98, vol. 27, pp. 94–105 (1998)

5. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is nearest neighbor meaningful? In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 217–235. Springer, Heidelberg (1998)
6. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: massive online analysis. J. Mach. Learn. Res. **99**, 1601–1604 (2010)
7. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: SDM' 06, pp. 328–339 (2006)
8. Chen, Y., Tu, L.: Density-based clustering for real-time stream data. In: KDD '07, pp. 133–142 (2007)
9. Ester, M., Kriegel, H.-P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD '96, vol. 96, pp. 226–231 (1996)
10. Goil, S., Nagesh, H., Choudhary, A.: MAFIA: efficient and scalable subspace clustering for very large data sets. In: KDD '99, pp. 443–452 (1999)
11. Hassani, M., Kim, Y., Seidl, T.: Subspace MOA: subspace stream clustering evaluation using the MOA framework. In: DASFAA' 13, pp. 446–449 (2013)
12. Hassani, M., Kranen, P., Seidl, T.: Precise anytime clustering of noisy sensor data with logarithmic complexity. In: SensorKDD '11 Workshop in conj. with KDD '11, pp. 52–60 (2011)
13. Hassani, M., Müller, E., Seidl, T.: EDISKCO: energy efficient distributed in-sensor-network k-center clustering with outliers. In: SensorKDD '09 Workshop in conj. with KDD '09, pp. 39–48 (2009)
14. Kailing, K., Kriegel, H.-P., Kröger, P.: Density-connected subspace clustering for high-dimensional data. In: SDM'04, pp. 246–257 (2004)
15. Moise, G., Sander, J., Ester, M.: P3c: a robust projected clustering algorithm. In: ICDM '06, pp. 414–425 (2006)
16. Patrikainen, A., Meila, M.: Comparing subspace clusterings. TKDE **18**(7), 902–916 (2006)