

Clustering Data Streams: A Complex Network Approach

Sandy Porto[1] and Marcos G. Quiles[2(✉)]

[1] National Institute for Space Research (INPE), São José dos Campos, SP, Brazil
sandyporto@gmail.com
[2] Institute of Science and Technology, Federal University of São Paulo (UNIFESP), São José dos Campos, SP, Brazil
quiles@unifesp.br

Abstract. Clustering data streams is an interesting and challenging problem. Although several solutions have been proposed in the literature, some drawbacks remain. For instance, how to deal effectively with the offline process for partitioning the micro-clusters into macro-clusters is still an open problem. Typically, the k-means algorithm is considered in this phase, which despite precise results, require a mandatory user-defined parameter k, that defines the number of expected clusters. In this paper, we propose a new clustering method for data stream, named Prototype Networks. This method takes the complex network structure to represent the set of micro-clusters. This approach has proven to be advantageous mainly because these networks have an inherent community structure. As a consequence, the offline phase might be easily handled by a community detection algorithm, such as Infomap. The communities detected represents the cluster structure of the data assuming that the network construction was designed for this purpose. Computer experiments demonstrated the feasibility of the proposed approach. Moreover, the proposed method can detect automatically the number of clusters in evolving scenarios, which is a useful feature when dealing with data streams with concept drift.

Keywords: Data streams · Clustering · Complex network

1 Introduction

Data streams are a concept of data in which elements are streamed live [2,6,13,14,21,22]. Due to its live nature, the volume of data might overcome the computational resources making it difficult to apply traditional machine learning techniques. Moreover, data patterns, such as clusters, can evolve through time, imposing an additional challenge when dealing with data streams [15,18,19,24,27].

Similarly to traditional machine learning scenarios, several tasks can be performed with data streams, such as data classification and data clustering [2,6]. Here we will focus on the data clustering problem, which might be roughly defined as the problem of partitioning the samples into groups of similar objects.

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11619, pp. 52–65, 2019.
https://doi.org/10.1007/978-3-030-24289-3_5

Several approaches have been proposed to deal with data clustering of data streams [14,21]. A critical restriction imposed on data stream algorithms is the one-pass constraint. It states that each sample must be analyzed only once by the algorithm. To overcome this limitation, algorithms have employed the strategy of dividing the task into two phases, named online and offline [1,4]. The online phase, which is conditioned to the one-pass constraint, is responsible for summarizing the incoming data. On the other hand, the offline one is accountable for analyzing the summary provided by the online phase to extract the partitions and detecting the changes into the data trend [2,3,6,29].

Considering the importance of the summary structure generated by the online phase and taken into account by the second, here, we investigate an alternative data representation via a complex network framework [11,12,20]. Explicitly, data prototypes are modeled as nodes and their similarities encoded into the links of the network. As a consequence, the generated network highlights the structure of the micro-clusters and their relations encoded in a powerful relational structure, a complex network.

A Complex Network posses many features that can be extracted from its structure; among them, there is a topological feature known as community structure that divides the vertices into groups according to their links density. Thus, these communities, once revealed by a community detection algorithm, might be interpreted as a clusters partition if the creation and maintenance of the network are designed for this purpose. In summary, we propose a clustering algorithm in which the online phase is responsible for generating the network of prototypes, or the summary of the samples, while the offline phase carries out the community detection process, or the clustering partition.

Three experiments were performed. The first two evaluate the clustering accuracy achieved by our approach, and the third one verifies whether the community structure of the network evolves revealing the natural cluster evolution existent in the data flow. Results showed that the communities in the network indeed follows the changes in the cluster pattern, demonstrating the capacity of the method for detecting changes in the data flow, such as drifts. Although revealing the evolution of the cluster structure on evolving data, our method delivers a slightly lower accuracy in comparison to benchmark approaches, such as the CluStream algorithm, which is still one of the drawbacks of our approach.

This paper is organized as follows. Section 2 presents the related work. Our proposed algorithm, named Prototype Network, is presented in Sect. 3. Section 4 presents our experiments and results. Finally, in Sect. 5, some conclusions are discussed.

2 Related Work

Nowadays, data have been generated in large quantities and the significant technological advance of the last two decades is directly related to this phenomenon [2,19,29,30]. Advances in hardware and software have allowed anyone to have

access to some sort of data flow even on a large scale [7,29]. To cite a few examples: social networks data, such as Facebook, Twitter, Waze, Uber, Google Maps; time series related to finance, actions and exchange, or associated with the processing of earth observation images can be used for almost anyone with some knowledge on how to manipulate it.

These data flows are known in the literature as a data stream, that is, a sequence of objects generated and made available automatically at high frequency through the most varied type of sources. Data streams have received much attention in recent years because manipulating, observing, mining, grouping, classifying and seizing this type of data is being demanded by a variety of organizations [3–5,31].

However, this availability of data has no practical value if it cannot be intelligently transformed into meaningful information [7]. These flows need to be stored and analyzed in some way, even though they have a different kind of restrictions, for instance, a finite set of resources, particularly processing time and memory usage [2,7,29,31]. Beyond difficulties discussed above, data streams have yet even more challenges, for example, the clustering of evolving data, i.e., how to cluster data that is fluid? In the next section, these challenges behind this task are highlighted.

2.1 Challenges of Clustering Data Streams

Firstly, from a computational point of view, a data stream F can be represented as a sequence of objects $x^1, x^2, \cdots, x^N$, such that $F = \left\{x^i\right\}_{i=1}^{N}$, and $N \to \infty$ because that sequence is potentially unlimited. Secondly, each object x^i has intrinsic to itself a time stamp i, which can be assumed as a primary key. Thirdly, the object x^i is a vector of attributes of dimension n, such that $x^i = \left[x_j^i\right]_{j=1}^{n}$. Fourthly and finally, the value of each attribute x_j^i can be continuous, categorical or mixed [8,15,29,31].

Therefore, due to time constraints, it is necessary that the objects from the data streams can be processed, analyzed and the model possibly updated as fast as the arrival rate of the data [5,19]. In addition, space constraints make it unworkable or impractical to store a copy of each received object. In this way, an abstraction step can be used as a way of cutting the data into relevant attributes [7,31].

After the abstraction data phase, the result needs to be processed, analyzed and compared, so the model can decide what to do with the sample. The object can be discarded or incorporated into the model, either way, the decision, and possibly update should perform faster than the arrival of new data. For comparison, in traditional data analysis, a representative or decision model is usually created after analyzing the data in its entirety more than once [4,8,19]. In the context of data streams, it is also likely that the model has to keep at least some meta-data (timestamp, for example) in this process, as these can contribute to the detection of concept drifts.

In short, the large amount of data flowing into the system cannot be stored in its entirety. Thus, their contribution to the model construction and update is minimal. However, in some cases, if the tools mentioned above, the collection of meta-data and the abstraction steps, combined with an adequate structure to keep an information summary of data, it is possible to create a framework which can lead to near-optimal results. The structure used in this work is inspired by Micro-Clustering approach used by Aggarwal *et al.* in CluStream Algorithm [1,2,4] which are detailed in the next sections.

2.2 Micro-clustering Approach

The summary structure has to be concise enough for updating model be efficient and, at the same time, to contain temporal and space data enough to ensure an accurate cluster partition through time. Those constraints are difficult to achieve and raise the following questions: (a) the structure has to save all data ever received or will erase information identified as imprecise or old? If it decides to delete information, in which moments in time? and (b) how that same structure can be the model that will provide cluster partition and detect concept drift through time?

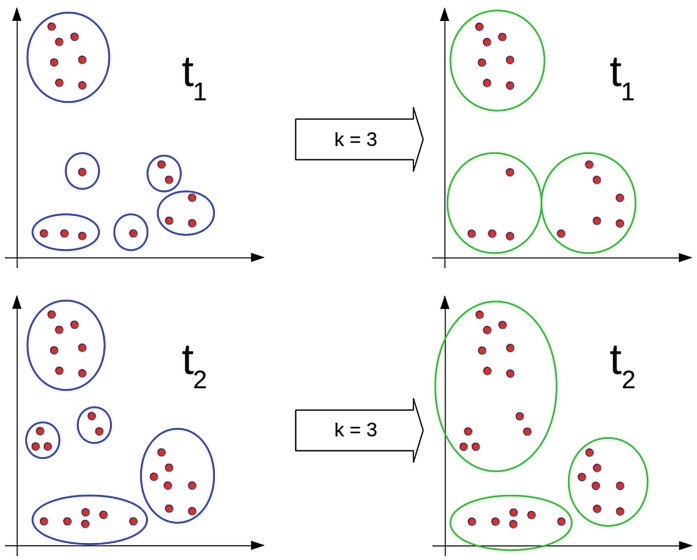


Fig. 1. Macro-cluster creation setting parameter k to 3 on CluStream.

CluStream uses a concept known as *micro-clusters* that are efficient to collect data from a data stream. The micro-clusters maintain statistical information about nearby samples in data space. CluStream process the data twice, first the

online phase is responsible for processing the arriving data and to alter the micro-clusters if necessary, and the offline component is responsible for analyzing the summary in order to obtain current cluster partition and to maintain an index which will tracking changes in relevant aspects to detect concept drift. In this scenario, the micro-clusters set composes the *summary structure*.

Comparison of the results when applying k-means on micro-cluster's state is also useful to detect major changes in data. In Fig. 1, the macro-clusters, which are the result of applying k-means ($k = 3$) on micro-clusters at time t_1 and time t_2, could give insights on how data changes.

One problem of using k-means is the mandatory user-defined parameter k. After all, if the k value could be discovered alongside the clusters themselves, thus the clustering would become parameter-free. This paper proposes a summary structure that besides keep micro-clusters information, also keeps information about how much these micro-clusters get together or split out trough time. That extra information provides inputs for clustering process that makes unnecessary to predetermine parameter k's value, even though costs accuracy and effectiveness. Nevertheless, that could be used in some context.

The algorithm proposes in this paper stores the summary in a structure inspired by Complex Networks. These networks have a property known as *community structure*, that could be interpreted as a representation of data partition in clusters. For that, one has to perform some community detection algorithm in the network, for example, Infomap, which do not need to predetermine the value of parameter k [25,26].

3 Prototype Network

This paper proposes a novel method based on a complex network structure. The method, named Prototype Network, or PN, considers two sets: vertices $V = \{v_1, v_2, \cdots, v_n\}$, and a set of edges $E = \{(v_i, v_j) \,|\, v_i \text{ and } v_j \text{ are neighbors}\}$, which jointly are represented as $PN = \{(V, E)\}$. Each vertex and edge are represented as list structures. Vertices are indexed by its integer primary key i and a tuple (i, j) is assigned to each edge.

The Prototype Network structure considered in this work contains a function that adds a new vertex that contains the object x^i to the network $PN = newVertex(PN, x^i)$. In this function the object x^i is loaded to a vertex v_i, added to the network and the network is updated if necessary, following the flowchart contained in the Fig. 2.

In the context of this paper, the Prototype Network will be used for clustering a Data Stream, so there are some restrictions on time and memory. The time constraint says that the update of the structure must be as fast as the arrival rate of new objects and the memory constraint says that the structure can not grow at the same pace as the data stream because this would lead to an unlimited structure.

Thus, when a new vertex is added to the network, the structure invokes the function that chooses the neighbors for it. The *chooseNeighborhood*() function

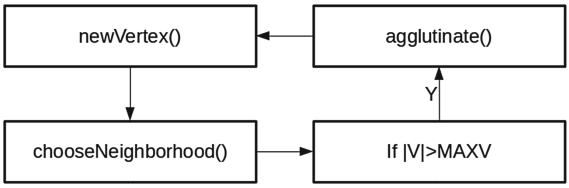


Fig. 2. The following steps after calling the *newVertex*() function. First, the *chooseNeighborhood*() function is invoked that choose which vertices already in the network will make pair with the new one, after that, it is verified if the size of the network, that is, the number of vertices becomes higher than parameter $MAXV$. If yes, the algorithm calls the *agglutinate*() function, which decreases the number of vertices by merging two of them into one and adding the newly merged vertex using the *newVertex*() function itself.

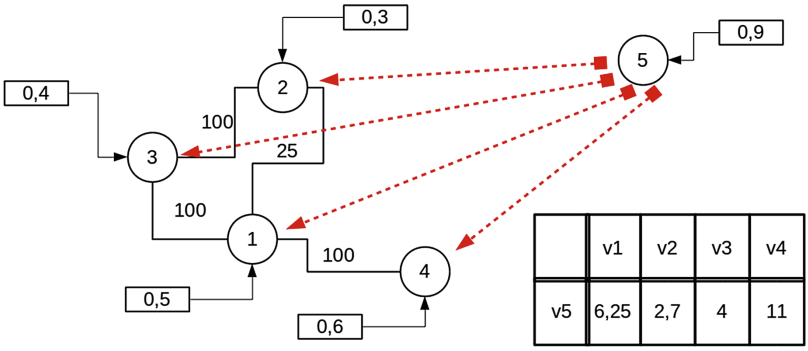
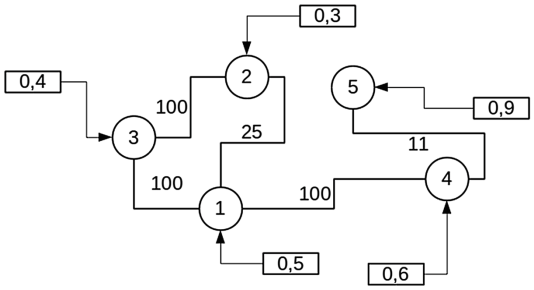


Fig. 3. Prototype Network before choose the neighborhood for vertex v_5. The network to the left possess four vertices and four edges, the vertices v_1, v_2, v_3, v_4, and the edges (v_1, v_3), (v_2, v_3), (v_1, v_2), (v_1, v_4), the value of object x for each vertex is represented in rectangles attached to the circles and the value of similarity of each edge is represented by numbers attached to the line. The red arrows show which pair of vertices have their similarity calculated. Finally, in bottom right, a table resuming the results of similarities. (Color figure online)

choose some existing vertices that will link to the new one. For each selected vertex, a new edge is created. Additionally, a user-defined parameter, called $MAXV$, tells the maximum number of vertices allowed in the network structure.

The function *newVertex*() is continually invoked until the number of vertices exceeds $MAXV$. When this occurs, the Prototype Network runs the agglutination method, which involves merging pairs of network's vertices in order to reestablish $|V| = MAXV$.

It can be seen that there are two crucial steps to prototype network algorithm: neighborhood choice (Figs. 3 and 4) and agglutination (Figs. 4 and 5). The choice of the neighborhood occurs when a new vertex is added, the data sample loaded into the vertex will be compared to the data stored in the other vertices of the



(v1,v3)	100
(v1,v4)	100
(v2,v3)	100
(v1,v2)	25
(v4,v5)	11

Fig. 4. Resulting Prototype Network after choosing neighborhood for vertex v_5 and showing all current edges on the network.

network, one by one, as shown in Fig. 3. Here, the comparison between patterns is performed via a similarity measurement based on the Euclidean Distance, as in Eq. 1.

$$sim(v_i \rightarrow x, v_j \rightarrow x) = 1/dist(v_i \rightarrow x, v_j \rightarrow x) \tag{1}$$

After this calculation, a cut-off strategy is used to define the neighbors of the new vertex, the cutoff is established by choosing only those vertices that have greater similarity than the mean plus standard deviation of all computed similarities. For example, in Fig. 3, the similarities obtained are 6.25, 2.7, 4 and 11, so the mean is approximately 6 and the standard deviation is approximately 3.6, so the threshold value is $6 + 3.6 = 9.6$, all possible connections in which the similarity are below this threshold are discarded. Therefore, in this example, only the v_4 will be assumed as a neighbor of new vertex v_5. The weight of that edge is set accordingly to the similarity between these nodes, that in this example is 11.

The other crucial step, the *agglutinate*() function, examines the set of edges and verifies those that have greater weight. Assuming that $MAXV$ parameter value for the network in Fig. 4 is 4, some edge, and therefore a pair of nodes will be agglutinate (merged). As explained previously, this weight is proportional to the similarity between the objects stored in the vertices. In the network presented in Fig. 4, the possible values are shown in the table depicted in this figure.

Assuming, that edge (v_2, v_3) is selected, the two vertices v_2 and v_3 are submitted to the following process: it saves the objects information and delete the two vertices of the network, merges the information of the two objects (i.e. compute the centroid) and adds a new vertex with the resulting object represented by the new centroid. This process is presented in Fig. 5.

The addition of the agglutinated vertex follows the same process of adding a new object, so it calls the function *newVertex*() that determine the neighbors using the function *chooseNeighborhood*(), and, if necessary, performs the agglutination method again, as reproduced in Fig. 5. The final result of vertex v_5 addition can be seen in Fig. 5.

Finally, to perform the partition of the network and reveal the clusters, a community detection method is applied, which divides the vertices into groups according to the edge's densities. It is worth noting that each vertex represents

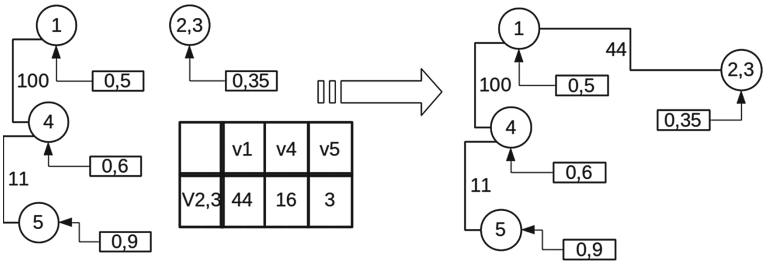


Fig. 5. Final Result Network after choose the neighbourhood for vertex v_5 and performed the *agglutinate*() function.

a prototype of the sample, or a micro-cluster, thus, more than one object is assigned to it. Thus, if vertex v_i is determined as part of community C^i, all objects represented by v_i will be considered belonging to the same group of all other objects of vertices of C^i. Although any community detection method could be taken into account, here, we adopted the Infomap algorithm [25,26], due to is higher efficiency and absence of parameters.

4 Experiments

Three experiments were performed to evaluate the Prototype Network. The first two evaluates the accuracy of the clustering process and compares its outcome with the results provided by the CluStream algorithm. The third experiment checks if the number of communities in the structure follows the change in the number of clusters of the data as the data stream evolves. Eventually, the data stream experiences a change in the number of groups, which might increases or decreases. However, detecting the emergence of a new community is especially difficult as time is needed to rule out the hypothesis that arriving new data are not just noises.

It should be stated that the CluStream algorithm was adopted as a competitor in our experiments due to the following reasons: (1) The CluStream is considered a benchmark algorithm in the literature; (2) it is on of the most cited methods for clustering data streams; and (3) both CluStream and Prototype Network use the micro-clustering approach.

All the experiments treated in this work were carried out on a PC with an Intel i3-6100 3.70 GHz processor with 4 cores and 8 Gb of RAM running Ubuntu 16.04.5 LTS 64-bit Operating System. The entire development of the algorithm was performed in R [23] in RStudio version 1.1.456. The packages *igraph* [9] and *stream* [16,17] were used, respectively, for the treatment of complex network-like structures, and for the generation of synthetic data and implementation and execution of the CluStream algorithm used in the experiments.

The CluStream algorithm had its parameter M set to 200, that is, the structure consists of 200 micro-clusters, and its parameter h set to 1000, although

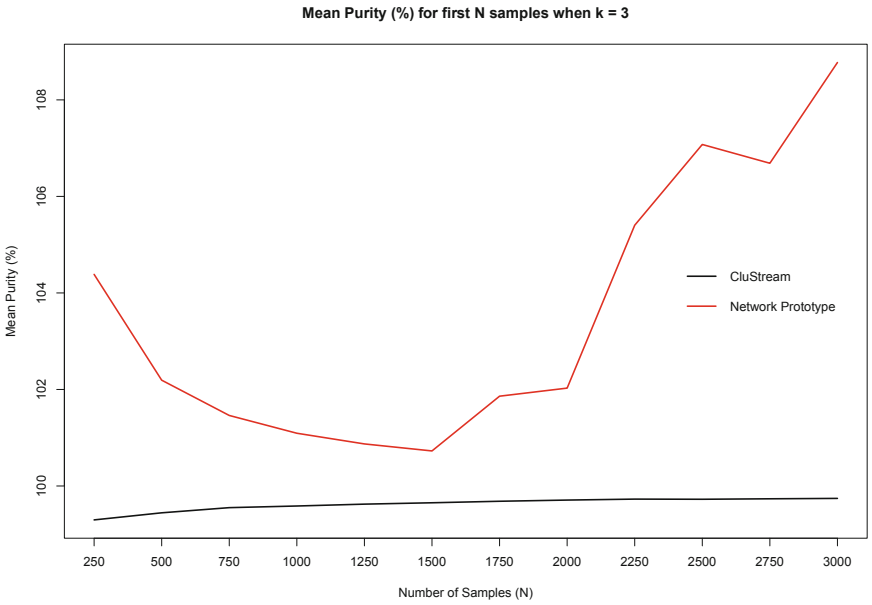


Fig. 6. First experiment result when $k = 3$. Mean Purity (%) value for the first N samples being N equal to 250 until 3000. (Color figure online)

these values do not influence results in this experiment. In the prototype network, the $MAXV$ parameter was also set to 200, that is, the network will consist of a vertex set of at most 200 vertices.

In order to execute the k-means for CluStream algorithm, the parameter k must be adjusted to be equal to the number of true classes. In the Infomap algorithm, the network itself is the only input parameter. The return result of each is compared to the true class and the purity of the partition in clusters is measured.

The value of the accuracy is calculated using the purity measure that verifies what percentage of the elements in each cluster belong to the dominant class. The dominant class of a cluster is the true class to which the majority cluster's elements belong. The purity measure is a positive value and best value is 100%, but results could be lower or higher than that, if the number of clusters n_C is equal or lower than k, than the purity will assume a value between 0 and 100, in the other case, if number of clusters n_C is higher than k, but two or more of clusters actually belongs to the same true classes, than purity will be higher than 100%.

Considering that $|C_i|$ is the size of cluster C_i, and $|C_i^d|$ is quantity of elements which belong to dominant class d, than purity could be calculated according to Eq. 2.

$$purity = \frac{\sum_{i=1}^{n_c} \frac{|C_i^d|}{|C_i|}}{k} \times 100\% \tag{2}$$

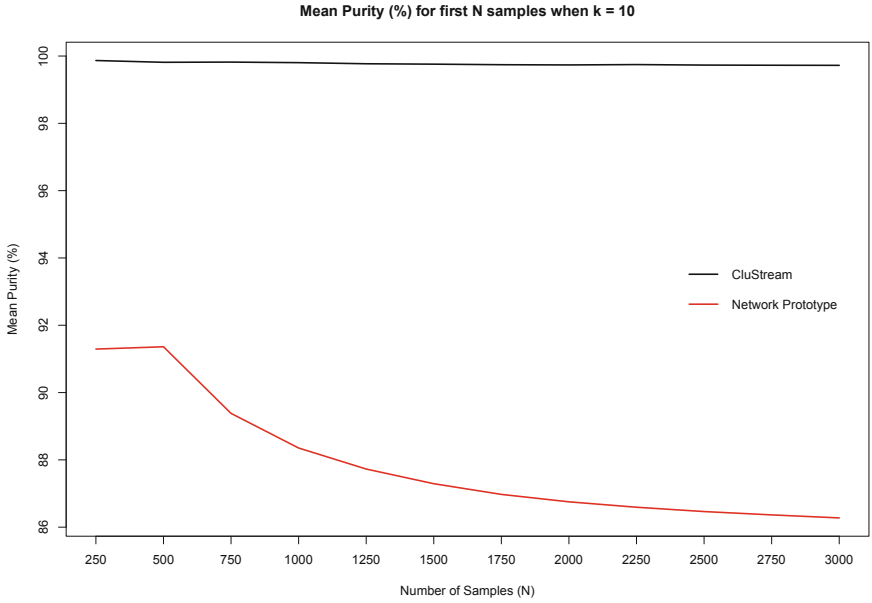


Fig. 7. First experiment result when $k = 10$. Mean Purity (%) value for the first N samples being N equal to 250 until 3000. (Color figure online)

The first experiment uses a ten-dimensional data stream with $N = 3000$ samples generated by the package *stream*. Each arriving object is analyzed and inserted into the structure, either the structure of the micro-clusters or the prototype network. After that, the k-means is applied to the micro-clusters and the Infomap community detection algorithm to the prototype network. Two versions of this experiment were analyzed: (1) when the data generated were divided into three clusters, thus $k = 3$, and (2) with ten clusters, $k = 10$.

The results can be seen in Figs. 6 and 7, both shows the evolution of purity mean measured every time the data stream received 250 samples, that is, the graph shows the mean value of purity for the first N results for clusters partition for each algorithm when the number of samples N was equal to 250, 500, 750 and so on until reach 3000 samples.

Because the CluStream has parameter k fixed, purity measured will always be between 0 and 100, and the algorithm achieves 100% most of the time in both versions of the experiment, $k = 3$ and $k = 10$. Prototype Network, on the other hand, might achieve mean values higher than 100% when $k = 3$, which indicates that, occasionally, $n_C > k$, i.e. two or more clusters belongs to the same correct class. When $k = 10$, Prototype Network mean purity value decay over time reaching 85%.

The second experiment uses a database available in the UCI repository [10], called Vehicle Silhouettes [28]. The eighteen attributes of this data set are numbers extracted from the processing of 846 different images that have silhouettes of four different types of vehicles. Therefore, the database has 846 instances,

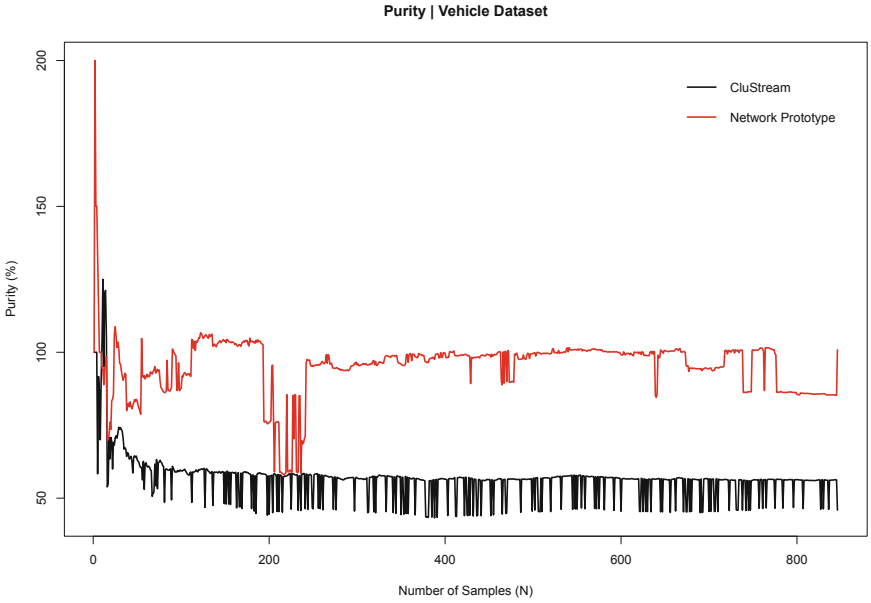


Fig. 8. First experiment result for Vehicle Data set. The graph shows purity value for CluStream and Prototype Network after arrival of each new sample. (Color figure online)

$N = 846$, 18 dimensions, and four classes. Although the database is normally interpreted in its entirety in traditional algorithms, in this experiment it was adapted to imitate the behavior of a data stream, that is, each instance is presented separately to the model.

The results of this experiment show the evolution of the purity value with each introduction of a new object for both CluStream and Prototype Network, and can be seen in Fig. 8. The graph shows that, in this case, the Prototype Network algorithm obtained even better results than CluStream. In addition, it is possible to observe in this experiment that Prototype Network algorithm had better results in the discovery of the number of clusters, since it has a purity value less than or equal to 100% most of the time.

Next, the third experiment evaluates whether the number of communities follows automatically the number of clusters in data stream. In special, when new clusters are introduced to the algorithm. This experiment also considered data generated by package *stream*. However, the data is reorganized so that groups of classes are put together in batches that will be presented to the algorithm. The goal is to simulate the arrival of several new clusters in the data stream at certain moments in time and, without disregarding the old data, verify if the community structure also increases the number of communities, following the movement of the clusters.

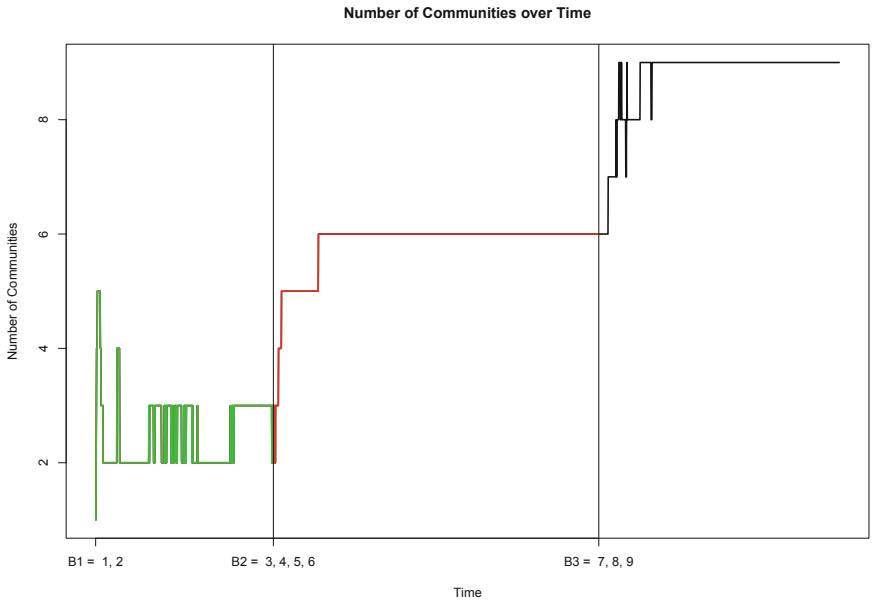


Fig. 9. Second experiment results. Shows how number of communities changes as new samples of each batch are introduced to the Prototype Network Structure. (Color figure online)

This experiment assume a ten-dimensional data stream with $N = 3000$ samples. The data is divided into nine true classes and three batches $B_1 = \{1, 2\}$, $B_2 = \{3, 4, 5, 6\}$ and $B_3 = \{7, 8, 9\}$, in total batch B_1 has 717 elements, batch B_2 has 1313 elements, batch B_3, 970, summing up the 3000 samples. The order of the elements is as follow, at first, only elements of batch 1 appear in the data stream but at random order within the batch, followed by the elements of batch 2, and then batch 3, also at random order within each other.

Figure 9 shows the number of communities in the network structure at each processing of a new object. Initially, the structure still are in construction phase, but eventually stable in 2 or 3 communities, although the number of true classes was just 2 for batch B_1, than when batch B_2 is introduced, rapidly the network adjust itself and stable in six communities which is the number of true classes too. Finally, batch B_3 is introduced, and after a transient, it reaches the same number of true classes correctly.

5 Conclusions

Developing methods able to deal with evolving data stream is still a big challenge. Here, we have proposed a novel method, named Prototype Network (PN). The PN provides a new manner to summarize the data into micro-clusters, prototypes, represented as vertices in a network structure. The edges between vertices

represent the similarity between prototypes. The partition of the network is performed by a community detection method. Although any community detection method might be taken into account, we have taken the Infomap into account due to its higher accuracy and lower time complexity.

Experiments showed that the accuracy delivered by the proposed method is competitive, albeit inferior to the results achieved by the CluStream. Even though, as illustrated in the third experiment, the Prototype Network can automatically detect and follow the evolving clustering structure of the data, which is a desirable property in clustering methods for data stream.

As future work, we intend to improve the dynamics of the PN to enhance its accuracy. Moreover, experiments with real streaming data sets will also be conducted.

Acknowledgments. This research was supported by the Brazilian National Research Council (CNPq Proc. 310908/2015-9 & 434886/2018-1) and the São Paulo Research Foundation (FAPESP Proc. 2011/18496-7 & 2015/50122-0).

References

1. Aggarwal, C.C.: A framework for diagnosing changes in evolving data streams. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, SIGMOD 2003, pp. 575–586. ACM, New York (2003)
2. Aggarwal, C.C.: Data Streams: Models and Algorithms. Advances in Database Systems. Springer, Heidelberg (2006). https://doi.org/10.1007/978-0-387-47534-9
3. Aggarwal, C.C.: An Introduction to Data Streams. In: Aggarwal, C.C. (ed.) Data Streams. Advances in Database Systems, vol. 31, pp. 1–8. Springer, Boston (2007). https://doi.org/10.1007/978-0-387-47534-9_1
4. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: Proceedings of the 29th International Conference on Very Large Data Bases, VLDB 2003, vol. 29, pp. 81–92. VLDB Endowment (2003). http://dl.acm.org/citation.cfm?id=1315451.1315460
5. Aggarwal, C.C., Yu, P.S.: On clustering massive text and categorical data streams. Knowl. Inf. Syst. **24**(2), 171–196 (2010)
6. Amini, A., Wah, T.Y., Saboohi, H.: On density-based data streams clustering algorithms: a survey. J. Comput. Sci. Technol. **29**(1), 116–141 (2014)
7. de Andrade Silva, J., Hruschka, E.R., Gama, J.: An evolutionary algorithm for clustering data streams with a variable number of clusters. Expert Syst. Appl. **67**, 228–238 (2017)
8. Barddal, J.P., Gomes, H.M., Enembreck, F.: SNCStream: a social network-based data stream clustering algorithm. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, SAC 2015, pp. 935–940. ACM, New York (2015)
9. Csardi, G., Nepusz, T.: The igraph software package for complex network research. InterJournal Complex Syst. **1695**, 1–9 (2006). http://igraph.org
10. Dua, D., Graff, C.: UCI machine learning repository (2019). http://archive.ics.uci.edu/ml
11. Estrada, E.: The Structure of Complex Networks: Theory and Applications. Oxford University Press, Oxford (2011)

12. Costa, L.F., Rodrigues, F.A., Travieso, G., Boas, P.R.V.: Characterization of complex networks: a survey of measurements. Adv. Phys. **56**, 167–242 (2007)
13. Gaber, M.M., Zaslavsky, A., Krishnaswamy, S.: A survey of classification methods in data streams. In: Aggarwal, C.C. (ed.) Data Streams. Advances in Database Systems, vol. 31, pp. 39–59. Springer, Boston (2007). https://doi.org/10.1007/978-0-387-47534-9_3
14. Gama, J.: A survey on learning from data streams: current and future trends. Progress Artif. Intell. **1**(1), 45–55 (2012)
15. Guha, S., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams. In: Proceedings of the 41st Annual Symposium on Foundations of Computer Science, FOCS 2000, p. 359. IEEE Computer Society, Washington, DC (2000). http://dl.acm.org/citation.cfm?id=795666.796588
16. Hahsler, M., Bolaños, M., Forrest, J.: Introduction to stream: an extensible framework for data stream clustering research with R. J. Stat. Softw. **76**(14), 1–50 (2017)
17. Hahsler, M., Bolanos, M., Forrest, J.: Stream: infrastructure for data stream mining (2018), R package version 1.3-0. https://CRAN.R-project.org/package=stream
18. Haidar, D., Gaber, M.M.: Data stream clustering for real-time anomaly detection: an application to insider threats. In: Nasraoui, O., Ben N'Cir, C.-E. (eds.) Clustering Methods for Big Data Analytics. USL, pp. 115–144. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-97864-2_6
19. Hyde, R., Angelov, P., MacKenzie, A.: Fully online clustering of evolving data streams into arbitrarily shaped clusters. Inf. Sci. **382–383**, 96–114 (2017)
20. Newman, M.: Networks: An Introduction. Oxford University Press, Oxford (2010)
21. Nguyen, H.L., Woon, Y.K., Ng, W.K.: A survey on data stream clustering and classification. Knowl. Inf. Syst. **45**(3), 535–569 (2015)
22. Qin, Y., Sheng, Q.Z., Falkner, N.J., Dustdar, S., Wang, H., Vasilakos, A.V.: When things matter: a survey on data-centric Internet of Things. J. Netw. Comput. Appl. **64**, 137–153 (2016)
23. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna (2018). https://www.R-project.org/
24. Rodrigues, P.P., Araújo, J., Gama, J., Lopes, L.: A local algorithm to approximate the global clustering of streams generated in ubiquitous sensor networks. Int. J. Distrib. Sens. Netw. **14**(10) (2018). https://doi.org/10.1177/1550147718808239
25. Rosvall, M., Axelsson, D., Bergstrom, C.T.: The map equation. Eur. Phys. J. Spec. Top. **178**(1), 13–23 (2009)
26. Rosvall, M., Bergstrom, C.T.: Maps of random walks on complex networks reveal community structure. Proc. Nat. Acad. Sci. **105**(4), 1118–1123 (2008)
27. Shao, J., Tan, Y., Gao, L., Yang, Q., Plant, C., Assent, I.: Synchronization-based clustering on evolving data stream. Inf. Sci. (2018)
28. Siebert, J.P.: Vehicle Recognition using rule based methods. TRM, 87-018, Turing Institute (1987)
29. Silva, J.A., Faria, E.R., Barros, R.C., Hruschka, E.R., Carvalho, A.C.P.L.F., Gama, J.: Data stream clustering: a survey. ACM Comput. Surv. **46**(1), 13:1–13:31 (2013)
30. Xu, J., Wang, G., Li, T., Deng, W., Gou, G.: Fat node leading tree for data stream clustering with density peaks. Know. Based Syst. **120**(C), 99–117 (2017)
31. Zhou, A., Cao, F., Qian, W., Jin, C.: Tracking clusters in evolving data streams over sliding windows. Knowl. Inf. Syst. **15**(2), 181–214 (2008)