An Online Incremental Clustering Framework for Real-Time Stream Analytics

Carlos Salort Sánchez*, Radu Tudoran†, Mohamad Al Hajj Hassan‡, Stefano Bortoli§,
Götz Brasche¶, Jan Baumbach‖, and Cristian Axenie**
Huawei German Research Center , *Huawei Technologies Duesseldorf GmbH* , Munich, Germany
Email: *carlos.salort@huawei.com, †radu.tudoran@huawei.com, ‡mohamad.alhajjhassan@huawei.com,
§stefano.bortoli@huawei.com, ¶goetz.brasche@huawei.com, **cristian.axenie@huawei.com
TUM School of Life Sciences Weihenstephan , *Technical University of Munich (TUM)* , Freising, Germany
Email: *salort@wzw.tum.de, ‖jan.baumbach@wzw.tum.de

***Abstract*—With the evolution of data acquisition methods, our ability to collect real time data has increased. This requires the development of real-time analytics, using the most recent data to generate valuable insights. One example is customer profiling, where we want to identify groups of similar clients who were active recently, and improve the quality of the suggestions. Traditional clustering algorithms perform well on finite datasets, but their execution is often not compatible with real-time requirements, especially for rapid changing trends. In this context, we propose a novel approach for the definition of incremental clustering algorithms to work within real-time constraints, in an online fashion, while preserving accuracy. We show the general applicability of the framework by employing this method to three different clustering algorithms. We compare the experimental results between traditional and online approaches evaluating accuracy and computational cost. The results show that algorithms executed in our framework are comparable to their offline implementation in terms of accuracy and with a high gain in execution time, up to three orders of magnitude on average.**

***Index Terms*—Data Stream Clustering, Online Clustering, Online Learning, Data Stream**

I. Introduction

In recent years, the amount of techniques that have been developed for analyzing data has greatly increased, correlated with the amount of data to process. This is driven by the volume, velocity, and, of course, the value that the data has to enable complex analytics in many verticals. Such new technologies require: 1) an increasing amount of data in short time, 2) the development of more complex algorithms, and 3) a special treatment of data that is continuously generated (i.e. data streams). Data streams can be seen as stochastic processes in which events occur continuously. There are some critical aspects characterizing streaming data that need to be addressed when developing algorithms. Data is assumed to flow continuously, possibly describing an unbounded dataset, and is typically non-stationary. Therefore the algorithm must judiciously use the resources for storing (i.e. memory) and processing the data in an efficient way (i.e. latency and throughput bounds) while the stream progresses. Moreover, such an algorithm must be flexible and adaptive to changes in data distribution, because the trends of the stream can evolve in time. In such a context, to meet the execution constraints, computation needs to be performed in an incremental fashion. All these requirements have paved the way for what recent years have witnessed an increasing interest: incremental learning on streams [2], [27], [17], [15], [20], incremental support vector machines [8], incremental neural networks [12], and incremental Bayesian models [26], respectively.

Traditional learning algorithms are not directly applicable in a streaming context, due to the different properties of the data generating process (i.e. non IID data) or the inherent changes in the data distribution (i.e. concept changes and drifts). Despite the fact that clustering is a technique that has been widely used for analyzing large datasets, there are only initial tryouts to explore stream versions of such algorithms [1], [4]. Their static or batch clustering versions are focused on static data, i.e., a static dataset that does not change its properties with time.

In this work we propose a framework to translate clustering algorithms to operate on streaming data. We enhance the benefits of clustering algorithms, by employing an efficient data orchestration and processing paradigm (i.e. stateful updates with process functions) [5], thus allowing them to operate in the streaming context. The proposed method comes as an alternative to well established approaches, such as the Massive On-line Analysis (MOA), in which all previous algorithms were integrated [2]. As a new paradigm, our approach does not try to optimize the execution of the classical algorithms on distributed systems, [13], [22], [14], rather it analyzes the decoupling between the model structure and parameters, such that we gain flexibility over the traditional approaches.

The paper is an initial instantiation of our framework. In section II we analyze previous work in stream clustering algorithms. Section III introduces our approach, and the methodology of translating traditional clustering algorithms to their streaming versions using our framework. We then introduce our preliminary experiments in section IV where we analyze the core advantages of our framework and confirm them through the thorough evaluation of our system. In section V we emphasize the contributions of the paper.

978-1-7281-4550-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ICMLA.2019.00243

II. Related Work

There are several works surveying streaming clustering algorithms [24], [19]. We will explain some of the most relevant algorithms. Previous efforts to design streaming[1] clustering algorithms are based on a two-phase process. An online phase of the algorithm responsible to update a structure that acts as a proxy of the underlying data stream, and an on-demand phase responsible for the actual cluster computation. CluStream [1] is an online clustering technique which employs an online phase and an offline phase alternatively as the stream progresses. The online phase creates a proxy of the real state of the stream events, reducing the amount of memory needed to store the data (i.e. a representation change). Internally, the algorithm uses a pyramidal system to reconstruct close events in the past. Another streaming clustering approach is DenStream [7]. It is also based of an online and an offline part. The online part works with potential micro clusters (i.e. p-micro-clusters) and outlier micro clusters (i.e. o-micro-clusters), which are stored in a different buffer. When receiving a new stream event, it first tries to join it with the closest p-micro-cluster (i.e. if it holds the cluster radius property). If this step fails, it trains the model to join with the closest o-micro-cluster, and confirm whether it changes into a p-micro-cluster. If previous operations fail, the algorithm puts the received event into a new o-micro-cluster. After a number of time steps, the algorithm re-calculates the weights and radius, and moves clusters from potential micro-clusters to outlier micro-clusters, or prune them. In another different approach [9] proposed D-Stream. The algorithm works in an online-offline way, using a grid and a decay factor for keeping the information in the online micro clusters. The assumptions it takes are proved not to affect the clustering results. This density based cluster works better than k-means for non-convex distributions shapes, and assumes an overall evolution of the clusters state over time. Another density-based online clustering algorithm is SDStream, [21] which was designed, opposite to the others, over sliding windows. It adopts a similar clustering framework, as CluStream (i.e. online micro-clustering - offline macro-clustering), and uses DBSCAN in the offline phase to create arbitrarily shaped clusters. Another algorithm of interest is the SOStream [16]. This is a fully-online clustering algorithm, in comparison to most of the other presented up until now. When SOStream receives a new stream event, it is assigned to the closer cluster. If the assignment is under a preset threshold, then neighboring clusters are drawn to the cluster. In the case clusters are overlapping, they are merged. In order to handle pruning or elimination of old clusters the algorithm uses a fading mechanism. In DENGRIS [3], they use sliding windows, and remove old data points when they expire. Then they cluster based on density regions. As one can observe, most of the algorithms in the literature work in a two-step fashion. Our approach differs from the state-of-the-art in that it is fully operating in streaming mode, i.e., there's no offline part, and the windows are strict.

[1]In the literature streaming algorithms are also known as online algorithms, as they operate in real-time as soon as the data arrives. In this work we use the terms interchangeably.

III. Incremental Clustering

As we saw in the previous section, there is an extensive research focused on streaming clustering algorithms. Most of the proposed algorithms do not work in a truly online-only fashion, as they need an offline processing step to collect the results. With our approach, we overcome that limitation by performing clustering in a continuous and incremental fashion. Another difference from previous work is the fading function, i.e. how much do previously processed events influence the clusters state. Previous work has been focused on using a decay function in order to forget old data. In our case, we use strict windows, which can be defined based on time (i.e. stream events collection time) or number (i.e. number of events). We do not carry any sort of influence from outdated data.

A. Developing incremental algorithms

The core contribution of this paper is a novel framework to convert clustering algorithms to operate in an incremental, online fashion, using two pieces of information: the clustering state at a current point in time and the computation to be applied on this state. This is the core element of our framework, namely the process function, which updates the state based on the previous state of the clustering, effectively reducing computation time by avoiding passing through all the data.

1) Clustering state: The state of the clustering is a standalone representation of all the stream events that are considered for clustering. This representation depends on the algorithm (i.e. events proxy in DenStream [7]). This state is altered by the process function (i.e. the process function reads the current state, process the current stream event, and overwrites to the current state).

2) Process function: The process function takes as input a clustering state and a new stream event, re-arranges the clusters (i.e. internal operations typical to a clustering algorithm), and returns the new state of the clustering. This step is where all the computations take place. This is the core advantage that our framework brings, by leveraging these computations with a data management and orchestration mechanism and stateful updates with process functions to reach the performance constraints under judicious resource allocation. The process function is triggered when a new data stream event is available, and it consists of a set of steps:

1) Modify the state in order to keep only the stream events that fulfill the conditions to be analyzed by the algorithm. Said conditions are given by the type of problem that is being addressed. Most common approaches include a fixed window of time t (only points generated in the last t seconds are evaluated), or a fixed amount of points p (only last p points are evaluated).
2) Update of the clusters state. The exact implementation will depend on the algorithm used. Clusters that lost points in the previous iterations, or the closer cluster to

the new data event will be updated. The key point to the update is to use the clustering state as the representation of the clusters for the calculations. This will reduce the time needed per iteration, as the algorithm does not run over the whole data stream.

3) Collection of results to fit the clustering state structure.

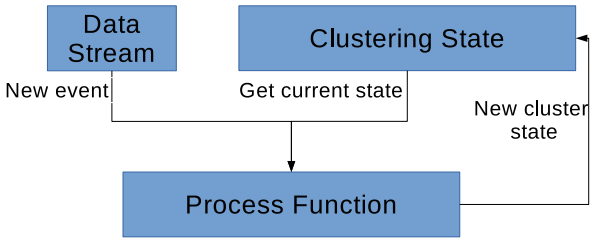

Fig. 1: Incremental clustering outline

3) Final result: After transforming the original clustering to operate with the two steps in our framework, the original algorithm is executed incrementally following the process described in Figure 1. Whenever new events are received, the process function receives them and simultaneously retrieves the current clustering state in order to, finally, update the clustering state to reflect the impact of the new event.

B. Sample instantiation

In order to evaluate the capabilities of our framework we have implemented three clustering algorithms: K-Means [18], Fuzzy C-Means [6], and Hierarchical Clustering [23]. We chose these three algorithms as they have a wide range of applications and methods, and at the same time to emphasize that our approach can be applied to different types of algorithms. We further explain the K-Means implementation to give an insight on how the framework can be utilized. K-Means is an iterative algorithm, which iterates over calculated centroids based in the average of the points that lie closer to the centroid. Our incremental implementation is described in algorithm 1.

Clustering state. The clustering state for the incremental K-means algorithm is defined by the centroids and the events contained into a cluster.

Process function.

- Input: Clustering state at the previous step, the new event received, and a set of parameters used for the algorithm that are defined at starting time. The parameters are the number of clusters k, and an iteration limit max-it.
- RemoveEvents: Remove old events (based on time, or number of events, depending on the configuration). Return the number of valid events in nr-events, and the clusters modified (because one of their points has been removed) in cl-to-adjust-e.
- FindCentroidMinDistance: Using a defined distance (euclidean distance in our implementation), find the closest centroid to the new event. Add the new event to the closest cluster, and update clustering-state. Finally, add the modified cluster to cl-to-adjust-c and cl-to-adjust-e.

```
Data: clustering-state, new-event, k, max-it
nr-events, cl-to-adjust-e =
 RemoveEvents(clustering-state);
if nr-events > k then
    clustering-state, cl-to-adjust-c, cl-to-adjust-e =
     FindCentroidMinDistance(new-event,
     clustering-state);
    lim = 0;
    while lim < max-it and cl-to-adjust-c > 1 do
        for centr in cl-to-adjust-c do
            clustering-state, cl-to-adjust-e =
             UpdateCentroids(centr);
        end
        Clear cl-to-adjust-c;
        for centr in cl-to-adjust-e do
            clustering-state, cl-to-adjust-c =
             ReAdjustEvents(centr)
        end
        lim++;
    end
else
    clustering-state = CreateNewCluster(new-event);
end
Result: clustering-state
```

Algorithm 1: Incremental K-Means process function

- UpdateCentroids: Updates the centroid calculating the average of the points in the given cluster, and updates clustering-state. Add modified clusters to cl-to-adjust-e.
- ReAdjustEvents: For each event in affected clusters, calculate distance to the other centroids. If the event is closer to a different centroid, it is modified in clustering-state. The affected cluster is added to cl-to-adjust-c.
- CreateNewCluster: If the number of valid events is smaller than k, then the new event is marked as a new cluster, and clustering-state is modified accordingly.

The advantage of re-writing the algorithm employing our framework is that we do not need to randomly initialize centers (as they are initialized in the warm up phase), leading to faster convergence. We only run over the whole data stream, in the worst case, with most of the times only accessing a small number of clusters.

IV. Evaluation

In this section we evaluate the incremental versions of the algorithms presented in section III. For K-Means, and hierarchical clustering, we use data streams generated from Gaussian distributions. For fuzzy C-Means, we use a different dataset, as we apply the algorithm to be used as a recommender system. We intend to show that following the approach proposed on the paper, similar results to the offline algorithms can be obtained, but with a much smaller time consumption per iteration.

A. Experimental setup

The experiments were executed on a single desktop machine with an Intel Xeon(R) E5-2630 v3 @ 2.40GHz 16 CPU, 32 GB RAM and Linux Ubuntu 16.04 operating system. We

monitored the execution with time probes (i.e. start-/end-time functions in Java). In terms of accuracy, we used two metrics: cluster purity and cluster overlap. Cluster purity is the extent to which clusters contain a single class. Given the ground truth partition $\{G_i\}$ of the data stream D, $|D| = n$, the clustering result $\{C_j\}$, and purity

$$P_j = \frac{1}{n_j} max_i(n_j^i),$$

then cluster purity is defined as

$$P = \frac{1}{n} \sum_j n_j P_j.$$

Purity is the number of objects in the cluster j that belong to cluster i. In other words, purity is the fraction of the cluster size that the largest class of objects that belong to that cluster represents. We use this measure to compare our results with ground truth. Cluster overlap is defined as a measure to mutually compare the results of two of clustering algorithms. It is the average of the purity of clustering algorithm A when assuming clustering algorithm B as the ground truth, and the purity of clustering algorithm B when assuming clustering algorithm A as the ground truth.

B. Evaluating the Streaming K-Means

To evaluate the implementation of incremental K-Means in our framework, we used two different data streams. The first one is the S1 dataset collected from [11], with a ground-truth of k=15, and 5000 points. The second data stream is an artificially generated one. It is also formed of 15 clusters, generated from various Gaussian distributions, and contains 150000 points. To evaluate the performance of the implementation in our framework, we used the first data stream to compare the purity of the algorithm. This dataset was selected because the ground truth for the number of clusters is known, and the performance of k-means should be high. We use the second dataset to compare execution time, as it is a similarly easy dataset for the K-Means algorithm, but it is much bigger, allowing for bigger windows.

We used the K-Means++ clustering algorithm modification of the K-Means algorithm [4] found in the Apache math java library - Machine Learning implementation.

For the time execution experiment we set k to 5, 10 and 15, and we chose the size of the windows to range from 50 to 20000 events. We plot the results in Figure 2. For the purity experiment, we used 5, 10 and 15 as k, the number of clusters. We set the size of the history window h, with values ranging from 50 to 2500 events in the history window. The results of this experiment can be seen in Table I. We compared our clustering results with the ground truth of the data stream.

The experiment procedure runs as follows: 1) for each combination of k and h, we run the clustering algorithm and we measure the time needed for an iteration of our incremental algorithm; 2) every h steps, we run the static algorithm, store the result, and measure the time (we also store the result for the incremental approach at the same time steps); 4) we compare

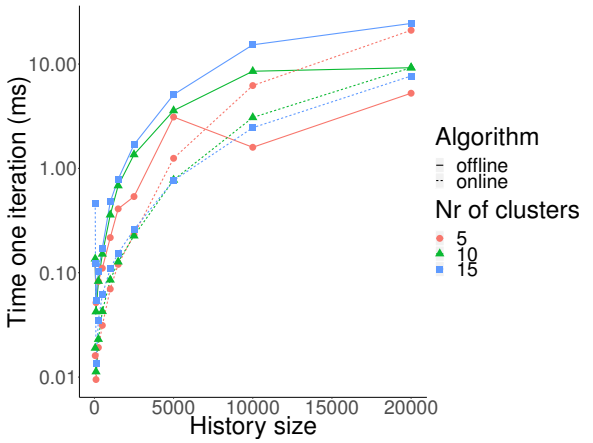

Fig. 2: Comparison between incremental and offline K-Means cluster purity between the static and the incremental algorithm and the average time needed for a run over the whole data stream.

		History size						
		50	100	250	500	1000	1500	2500
5	online	0.48	0.43	0.40	0.37	0.36	0.35	0.34
	offline	0.49	0.43	0.40	0.37	0.36	0.35	0.34
10	online	0.77	0.73	0.70	0.64	0.68	0.68	0.68
	offline	0.78	0.74	0.71	0.69	0.68	0.68	0.68
15	online	0.92	0.95	0.92	0.92	0.92	0.95	0.89
	offline	0.95	0.93	0.91	0.93	0.92	0.94	0.92

TABLE I: Average purity for different number of clusters and history sizes for both approaches.

In Figure 2, we can see that our incremental approach is around 3 times faster than the implemented K-Means++ for history size under 10000, and it gets closer as the parameter increases. This trend is maintained for clusters sizes 10 and 15, but k=5 is slower than the offline method after 5000 points in history. This speed-up proves that our implementation leverages the execution speed, but it also shows how optimized the K-Means++ algorithm is performing. The increase in execution time as the history size grows is explained by bigger memory usage, and because modifying one cluster is a more expensive operation (i.e. because on average clusters will be bigger). The increase in time for k=5 is explained by the bigger size in clusters, which make iterating over each of the clusters more expensive. In Table I, we see that for all clusters and history sizes, the purity of incremental and the offline K-Means++ clustering are similar. As a special case, we can see that the purity is higher for the $k = 15$ clustering, which makes sense due to the original data stream having 15 different clusters.

C. Fuzzy C-Means

We evaluated the implementation of the incremental Fuzzy C-Means, in a recommender system based on the data stream in [25]. This data stream gives information about the game users and contains user id, game, hours played, and whether the game was bought or not. The objective is to suggest the user a new game based on the assigned cluster. The ground truth is present in the datastream, in the case where

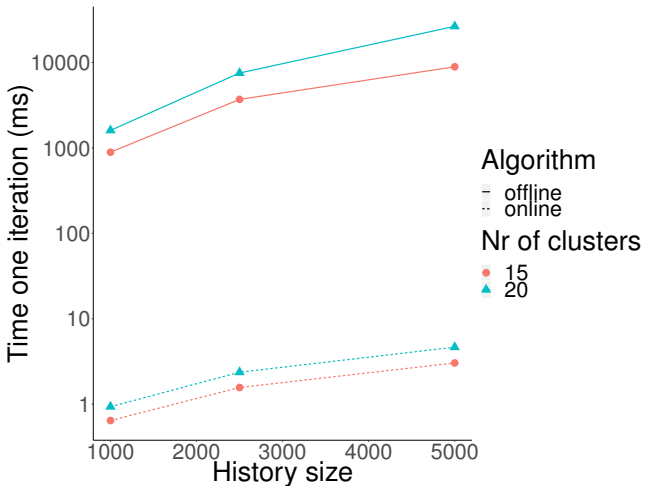

Fig. 3: Time comparison between Fuzzy C-Means implemented incrementally and basic implementation

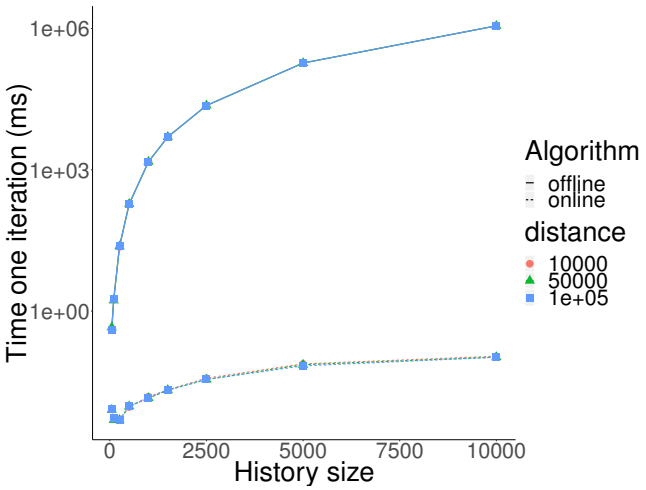

Fig. 4: Time comparison between incremental and online Hierarchical Clustering

the user also purchased the recommended game. We used as a basic algorithm the Fuzzy C-Means implementation from the Apache math - Machine Learning java library. We set parameters k to be 15 and 20 and h is set to be 1000, 2500 and 5000.

For each combination of k and h, we run the clustering. We measure the time needed for an iteration of our incremental algorithm. Every h steps, we run the static Fuzzy C-Means algorithm, store the result, and measure the time. We store the clustering result of the incremental algorithm, and the recommendations made. The first experiment compares the average time needed for the algorithm to finish an iteration (Figure 3). The second experiment checks the accuracy of the suggestions that the algorithm generated (Table II), whereas the third experiment compares how similar the clusters generated by both methods using the clustering overlap method (Table II).

Looking at Figure 3, using our framework the clustering algorithm has a 1000-times gain in speed for same history size, independently of the cluster size. Fuzzy C-Means is not as efficient as K-Means++ and we can see that in both cases it's slower than the K-Means implementation. Execution time goes up with history size due to increased complexity in the update operation. In Table II we show the quality of the suggestions that the algorithm generated, and the comparison between the results of the offline algorithm, and the incremental approach we proposed. We obtain a consistent accuracy in recommendation, independently of history size and number of clusters. This result shows that we can also support

		History Size		
	Nr. Clusters	1000	2500	5000
Good suggestions	15	0.841	0.862	0.859
	20	0.829	0.857	0.859
Cluster overlap	15	0.897	0.891	0.916
	20	0.895	0.882	0.891

TABLE II: Fuzzy C-Means quality comparison.Average values for different History sizes and Number of clusters

a fast recommender system, using only limited information from the complete data stream. We obtain similar results independently of history size and number of clusters, and the results are similar. These experiments prove that our clustering framework gives similar results to the offline algorithm, but with a large gain in execution time.

D. Hierarchical Clustering

We used the same two data streams we used for the K-Means experiment in subsection IV-B. We evaluated the clustering quality and the execution time. To compare the methods, we have implemented the algorithm presented in [10].

For this experiment, we choose the parameters history h, and distance d. The distance represents the maximum distances where two points should be clustered. Therefore, for bigger distances the clusters increase in size, whereas smaller distances generate smaller clusters in bigger quantities. For the overlap experiment, we chose history h to be ranging from 50 to 2500 points, and we chose distance d to be 5000, 10000 and 50000. This distances were selected to adapt to the data stream. For the execution time experiment we chose history ranging from 50 to 10000 points, and distances 10000, 50000 and 100000. Distances are bigger because the events in this data stream are more far apart.

Distance	History Size						
	50	100	250	500	1000	1500	2500
5000	0.999	0.998	0.994	0.986	0.971	0.962	0.955
10000	0.997	0.992	0.981	0.962	0.939	0.923	0.902
50000	0.98	0.963	0.942	0.932	0.93	0.939	0.949

TABLE III: Average Cluster overlap for different distances and history sizes between both approaches

For each combination of k and d, we run the algorithm. We measure the time needed for an iteration of our incremental algorithm. Every h steps, we run the static hierarchical clustering algorithm, store the result, and measure the time.

We store the clustering result of the incremental algorithm. Experiment one compares the average time needed for the algorithm to finish an iteration (Figure 4). Experiment two compares the clustering generated (Table III) using cluster overlap. We decided against cluster purity, since this method generates small clusters, which tends to increase purity. In terms of execution time (Figure 4) our incremental approach is much faster than the offline approach. The difference is considerable and consistent in history size. Distance influences the execution time just slightly. This is due to the increasing size of the data stream, as even with a bigger window, new data is not consistently close enough to increase significantly the number of iterations. Clusters from both approaches are consistently similar (Table III). The overlap seems to decrease with an increase in history size, which could be caused by the incremental method using approximations instead of exact values. The smaller distance gives more similar results. This is caused again by the increase in number of small clusters, which are easier to be similar. Our method gives similar results to the offline method, but is much faster.

V. Conclusions

We have presented an approach to transform clustering algorithms to operate on data streams in an incremental way, and we have applied it to three different algorithms. This comes as an effort towards truly streaming clustering and against dedicated modifications for certain scenarios. We have applied our framework to three widely used clustering algorithms (K-Means, Fuzzy C-Means and Hierarchical Clustering). We have proven that their incremental versions offer similar results to the original offline dataset-based algorithms they are inspired from, with an average purity of 0.85 (comparable to the offline algorithms), but with processing speed of up to an average of three orders of magnitude faster. We have seen that our approach is specially beneficial when comparing with a slower/less optimized algorithm, especially in real case scenarios (i.e. recommender system). Our future work with incremental algorithms aims at extending the palette of possible learning algorithms that work in a incremental fashion. We aim at converting density-based and graph-based algorithms into the incremental, streaming framework. We also aim to improve the efficiency of the converted algorithms, and tweak every individual instantiation to benefit of each individual clustering algorithm properties.

References

[1] Aggarwal, C., Han, J., Wang, J., Yu, P., J. Watson, T., Ctr, R.: A framework for clustering evolving data streams. VLDB (06 2003)

[2] Albert Bifet, Ricard Gavald, G.H., Pfahringer, B.: Machine Learning for Data Streams with Practical Examples in MOA. MIT Press (2018)

[3] Amini, A., Wah, T.Y., Teh, Y.W.: Dengris-stream: A density-grid based clustering algorithm for evolving data streams over sliding window. In: Proc. International Conference on Data Mining and Computer Engineering. pp. 206–210 (2012)

[4] Arthur, D., Vassilvitskii, S.: k-means++: The advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. pp. 1027–1035. Society for Industrial and Applied Mathematics (2007)

[5] Axenie, C., Tudoran, R., Bortoli, S., Hassan, M.A.H., Foroni, D., Brasche, G.: Starlord: Sliding window temporal accumulate-retract learning for online reasoning on datastreams. In: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA). pp. 1115–1122. IEEE (2018)

[6] Bezdek, J.C., Ehrlich, R., Full, W.: Fcm: The fuzzy c-means clustering algorithm. Computers & Geosciences **10**(2-3), 191–203 (1984)

[7] Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. vol. 2006 (04 2006). https://doi.org/10.1137/1.9781611972764.29

[8] Cauwenberghs, G., Poggio, T.: Incremental and decremental support vector machine learning. In: Proceedings of the 13th International Conference on Neural Information Processing Systems. pp. 388–394. NIPS'00, MIT Press, Cambridge, MA, USA (2000)

[9] Chen, Y., Tu, L.: Density-based clustering for real-time stream data. pp. 133–142 (01 2007). https://doi.org/10.1145/1281192.1281210

[10] Day, W.H.E., Edelsbrunner, H.: Efficient algorithms for agglomerative hierarchical clustering methods. Journal of Classification **1**(1), 7–24 (Dec 1984). https://doi.org/10.1007/BF01890115, https://doi.org/10.1007/BF01890115

[11] Fränti, P., Sieranoja, S.: K-means properties on six clustering benchmark datasets (2018)

[12] Furao, S., Ogura, T., Hasegawa, O.: An enhanced self-organizing incremental neural network for online unsupervised learning. Neural Networks **20**(8), 893–903 (2007)

[13] Gama, J.: Knowledge Discovery from Data Streams. Chapman & Hall/CRC, 1st edn. (2010)

[14] Gepperth, A., Hammer, B.: Incremental learning algorithms and applications. In: European Symposium on Artificial Neural Networks (ESANN) (2016)

[15] He, H., Chen, S., Li, K., Xu, X.: Incremental learning from stream data. IEEE Transactions on Neural Networks **22**(12), 1901–1914 (Dec 2011). https://doi.org/10.1109/TNN.2011.2171713

[16] Isaksson, C., Dunham, M.H., Hahsler, M.: SOStream: Self organizing density-based clustering over data stream. In: Perner, P. (ed.) International Conference on Machine Learning and Data Mining (MLDM'2012). pp. 264–278. Lecture Notes in Computer Science LNAI 7376, Springer (July 2012)

[17] Jayram, T., McGregor, A., Muthukrishnan, S., Vee, E.: Estimating statistical aggregates on probabilistic data streams. ACM Transactions on Database Systems (TODS) **33**(4), 26 (2008)

[18] Lloyd, S.P.: Least squares quantization in pcm. IEEE Transactions on Information Theory **28**, 129–137 (1982)

[19] Mansalis, S., Ntoutsi, E., Pelekis, N., Theodoridis, Y.: An evaluation of data stream clustering algorithms. Statistical Analysis and Data Mining: The ASA Data Science Journal **11**(4), 167–187 (2018)

[20] Popa, L., Budiu, M., Yu, Y., Isard, M.: Dryadinc: Reusing work in large-scale computations. In: Proceedings of the 2009 Conference on Hot Topics in Cloud Computing. HotCloud'09, USENIX Association, Berkeley, CA, USA (2009)

[21] Ren, J., Ma, R.: Density-based data streams clustering over sliding windows. In: 2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery. vol. 5, pp. 248–252 (Aug 2009). https://doi.org/10.1109/FSKD.2009.553

[22] Sayed-Mouchaweh, M., Lughofer, E.: Learning in Non-Stationary Environments: Methods and Applications. Springer Publishing Company, Incorporated (2012)

[23] Sibson, R.: SLINK: An optimally efficient algorithm for the single-link cluster method. The Computer Journal **16**(1), 30–34 (01 1973). https://doi.org/10.1093/comjnl/16.1.30

[24] Silva, J.A., Faria, E.R., Barros, R.C., Hruschka, E.R., De Carvalho, A.C., Gama, J.: Data stream clustering: A survey. ACM Computing Surveys (CSUR) **46**(1), 13 (2013)

[25] Tamber: Steam video games (2017), https://www.kaggle.com/tamber/steam-video-games

[26] Wilson, R.C., Nassar, M.R., Gold, J.I.: Bayesian online learning of the hazard rate in change-point problems. Neural Computation **22**(9), 2452–2476 (2010). https://doi.org/10.1162/NECO_a_00007

[27] Yang, J., Widom, J.: Incremental computation and maintenance of temporal aggregates. In: Data Engineering, 2001. Proceedings. 17th International Conference on. pp. 51–60. IEEE (2001)