Information Sciences Letters

Volume 9
Issue 3 *Sep. 2020*

Article 8

2020

Parallel outlier detection in real time data streams

Mohamed Sakr
Computer Science Dept. Faculty of Computers and Information, Menoufia University, Egypt,
walid.atwa@ci.menofia.edu.eg

Walid Atwa
Computer Science Dept. Faculty of Computers and Information, Menoufia University, Egypt\\ *College of Computing and Information Technology, Khulais, University of Jeddah, Saudi Arabia*,
walid.atwa@ci.menofia.edu.eg

Arabi Keshk
Computer Science Dept. Faculty of Computers and Information, Menoufia University, Egypt,
walid.atwa@ci.menofia.edu.eg

Follow this and additional works at: https://digitalcommons.aaru.edu.jo/isl

Recommended Citation

Sakr, Mohamed; Atwa, Walid; and Keshk, Arabi (2020) "Parallel outlier detection in real time data streams," *Information Sciences Letters*: Vol. 9 : Iss. 3 , Article 8.
Available at: https://digitalcommons.aaru.edu.jo/isl/vol9/iss3/8

This Article is brought to you for free and open access by Arab Journals Platform. It has been accepted for inclusion in Information Sciences Letters by an authorized editor. The journal is hosted on Digital Commons, an Elsevier platform. For more information, please contact rakan@aaru.edu.jo, marah@aaru.edu.jo, u.murad@aaru.edu.jo.

Inf. Sci. Lett. **9**, No. 3, 211-217 (2020)

Information Sciences Letters
An International Journal

http://dx.doi.org/10.18576/isl/090308

Parallel outlier detection in real time data streams

Mohamed Sakr[1], *Walid Atwa*[1,2,*] *and Arabi Keshk*[1]

[1]Computer Science Dept. Faculty of Computers and Information, Menoufia University, Egypt
[2]College of Computing and Information Technology, Khulais, University of Jeddah, Saudi Arabia

Received: 12 March 2020, Revised: 9 July 2020, Accepted: 13 July 2020
Published online: 1 Sep. 2020

Abstract: Outlier detection is one of the major problems in modern applications. Specially, detecting outliers for streaming applications, as data can dynamically change in subtle ways following changes in the underlying infrastructure. Due to the evolution in data in ratio of data generated every second and velocity, detecting outliers in these types of data becomes a very challenging task. This makes processing the whole data one time is impossible. In this paper we propose a parallel window based local outlier detection (PWLOD) algorithm that can detect outliers in real time using the sliding window algorithm and partition each window among several processing nodes. Each processing node process its portion of window using Local Outlier Factor algorithm and send the results to the master node which collects the results and process them to select the outliers. The experimental results show that the proposed algorithm has better execution time and accuracy than the state-of-the-art algorithms.

Keywords: Outlier detection, Data streams, Parallel processing.

1 Introduction

Anomaly detection or outlier detection has been an important problem in most data mining application. Specially, for streaming applications such as network intrusion detection. A lot of algorithms can detect anomalies in static data. As, the whole data exists, many iterations can be done over the data and the execution time doesn't have main focus as the data is processed offline. Recently, a huge amount of data is generated every second and in different velocities these data are called data streams. One of the important characteristics of data streams is that its size is infinite and is increased in different velocities. Which makes it impossible to process and store the whole data in memory using the static-based algorithms [1]. One of the techniques that is used in outlier detection is density based techniques [2]. Density based techniques have a great ability to detect outliers in different densities and dealing with nonhomogeneous densities datasets. Local Outlier Factor (LOF) is a density-based algorithm that deals with static data [3,4]. Because LOF needs a huge amount of memory to store the data to process. Specifically, for detecting outliers, LOF stores all the points of the data and its distances between the all points in the memory. Also, in any change in data by adding or deleting any points the LOF needs to be recalculated on the whole data set. Such these limitations of LOF, it can't be used with data streams as data streams size are infinite and data are changing over time as new points arrive.

To reduce the space complexity of LOF, author in [2] propose MiLof algorithm which stores a subset of the whole data by clustering old data. But, this algorithm degrades the outlier detection accuracy, as it clusters the old data using k-mean algorithm which doesn't preserve the density of the data [5].

To overcome the problems that faces detecting outliers in data stream, we propose a parallel window based local outlier detection (PWLOD) algorithm. The PWLOD algorithm works as follows: when new data arrives the master node divides these data into number of windows. each window is filled with number of points predefined as input to the algorithm. After that the master node partition every window into number of grids as [6] after the partitioning algorithm finishes, each partition is sent to one of the registered slave nodes. Each slave node calculates the LOF to the partition and sends the results o the master node. Finally, the master node removes all the points that have LOF greater than predefined threshold. The PWLOD algorithm make a redundant point between each window and also between each grid in every single

* Corresponding author e-mail: walid.atwa@ci.menofia.edu.eg

© 2020 NSP
Natural Sciences Publishing Cor.

window. Also, the PWLOD balances the load between all the slaves by sending every slave the same number of points to calculate the LOF for them.

The PWLOD algorithm solves the main problems of anomaly detection in data streams:

1. Process large amounts of data that have infinite number of points using the window and grids.
2. Process the LOF for all the points parallel in distributed environment.
3. Solves the problem of variety in velocity as it sends the data to the slaves to process them with configured window size.

The rest of the paper is organized as follows: Section 2 gives a review on the related work. Section 3 describes how the PWLOD works. Section 4 views the experimental results over real data sets. Finally, the conclusion is presented in section 5.

2 Related Work

There are many outlier detection algorithms that can detect ouliers in static data in which all the data points are available as whole and its size is pre-calculated [7,8,9]. However, streaming application has a large amount of data is being generated at high speed and volume. So, outliers need to be detected in limited time at one pass. Outlier detection algorithms in data streams can be categorized into three groups: distance-based, distribution-based and clustering-based [10,11,12].

In distance-based category we can detect outliers by measuring the distance between each point to the other points in the data set [13]. Many adaptions to this approach have been done to use it in data streams. In [14, 15] the authors proposed an algorithm deal with the sliding window model, where outlier queries are performed in order to detect anomalies in the current window. However, it has limited memory requirements and returns an approximate answer based on accurate estimations with a statistical guarantee.The authors in [16] enhanced this algorithm by using micro clusters that minimize the distance computations. Thus, reducing the time complexity and memory consumption needed.

In distribution-based categories outliers are detected by having a knowledge about the distribution of the data set and compare it with the new incoming data [17]. In references [18,19] authors used the Gaussian mixture model (GMM), where the dataset is equipped to a certain number of Gaussian distributions and the model having low computational resources, but most of them require parameters as inputs and they also assume a fixed distribution in dataset, that is not appropriate with streaming data.

Clustering based categories tries first to divide the data into clusters depend on the distribution of the data. Then, some algorithms mark the clusters with small number of points as outliers. other algorithms mark the points that are far from the clusters by a threshold as outliers. However, most of these algorithms were proposed to cluster the data sets rather than detecting outliers, e.g., [20,21,22] and authors in [23] tries to cluster high dimensional data streams. Authors in [24] generate histograms for the clusters in the streamed data which they used later for mining and also in detecting outliers.

In [25] the authors proposed an algorithm that works on outlier detection on data stream that is capable of determining if any point is outlier within any period of time. The more the time the available the more the accuracy of detection. In [26] the authors proposed an algorithm that try to learn from the history of the data stream to determine the normal behavior of the current period.

3 Parallel Window Based Local Outlier Detection (PWLOD)

In this section, we explain how can we detect oulilers using the proposed algorithm PWLOD. The PWLOD algorithm has four phases: preprocessing phase, partitioning phase, processing phase and detection phase as shown in figure 1.

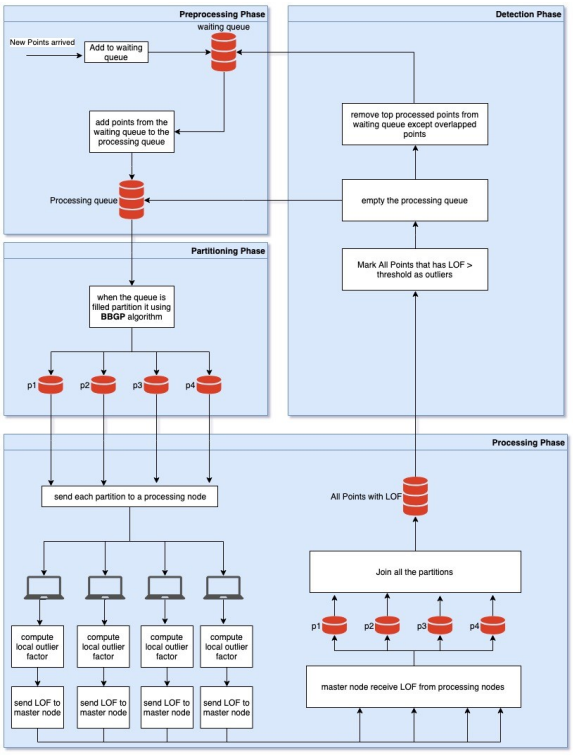

Fig. 1: System Architecture.

© 2020 NSP
Natural Sciences Publishing Cor.

Inf. Sci. Lett. **9**, No. 3, 211-217 (2020) / www.naturalspublishing.com/Journals.asp

3.1 Preprocessing Phase

In the preprocessing phase, when new bulk of points arrives in the stream, they will be added to a queue called waiting queue. When the number of the points in the waiting queue reaches processing window size *pws* (a predefined threshold) the PWLOD selects the top *pws* points from the waiting queue and sends them to the processing queue.

3.2 Partitioning Phase

When the processing queue becomes full the partitioning phase starts. The processing queue is partitioned into number of grids using the Balanced border-based Gird Partition algorithm (BBGP) in [6] as shown in algorithm 1. We can summarize BBGP algorithm as follows:

1. For every dimension in the dataset the BBGP algorithm finds the points that divide this dimension equally.
2. From all the generated points the BBGP algorithm calculate all the grids coordinates and add them to a grid set called *G*.
3. For each grid in *G* the BBGP algorithm calculates the new virtual boarder for it which is boarder percentage *B* * grid width.
4. All records that are located between the virtual and original boarder are duplicated between the two adjacent grids *g*
5. Sort all the grids in descending orders depending on the number of tuples in every grid.

After the BBGP algorithm finish its first step it then starts to distribute the grids to the nodes registered in the system. Every processing node is connected to the system it sends its information to the master node where the PWLOD register this node in its nodes catalog. The BBGP list all the processing nodes in the PWLOD catalog and assign each node with a grid as follows:

1. The BBGP loops through each grid and check if there is any node that didn't take any grid if yes it assigns that grid to this node.
2. If all the nodes have grids assigned to them, the BBGP algorithm calculates the average number of records in every node.
3. Then the BBGP algorithm selects all the nodes which have number of records less than or equal the average.
4. Next the BBGP algorithm assigns the grid to the node that has the number max of grids that are adjacent to this grid.
5. The BBGP algorithm repeats till it finish all the grids.
6. After the BBGP finish assigning every grid to every node, the BBGP algorithm initiates the processing phase.

Algorithm 1 Balanced border-based Gird Partition algorithm (BBGP)

input: processing node catalog *pnc* , number of grids in each dimension *gd*, dataset *D*, Border percentage *B*
output: Grids set allocated to slave nodes
1. Initialize $G = \{\}$
2. Divide all the records into grids such that each grid has the same number of records
3. Add all the grids into *G*
4. For each gird in *G* calculate its virtual boarder
5. For each grid *g* duplicate the records between the original boarder and the virtual boarder
6. Sort grids in *G* in descending order
7. **For** each grid *g* in *G* **do**
8. **if** there exist slave node with no grid **then**
9. Randomly choose a processing node with no grids and allocate *g* to it
10. **Else**
11. ε = the average number of records per processing node
12. Initialize a processing node set *N*' that contain all the slave nodes that has number of records less than or equal ε
13. *n* = select the processing node with the largest number of grids that are adjacent to *g*
14. Allocate *g* to *n*
15. **Endif**
16. **End**

3.3 Processing Phase

The processing phase loop through every record from the partitioning phase and encodes every chunk of data and send a processing request to every node with its predefined grids. Every node receives its chunk of data and decodes them to view all the grids then merge all the grids. Every node measure the LOF for all the records and send the results to the master node that can detect the outliers.

3.4 Detection Phase

The master node receives the results from every processing node and decodes them. Then it merges all the results with each other. the PWLOD algorithm loops through all the records and mark all the records that has LOF greater than threshold to be outlier. This threshold is predefined by the user. After that it removes all the points from the processing queue. Then it also removes top (window size * inter-window percentage) from the waiting queue. The inter-window percentage is predefined by the user to allow a connection between the windows to detect outliers between windows. For example, window w_1 process points 1,2,3 and 4 and window w_2 processes points 3,4,5 and 6 and so on as shown in fig. 2. Then the preprocessing phase starts to wait for the new arrival of the points to start again. The PWLOD algorithm is shown in Algorithm 2.

© 2020 NSP
Natural Sciences Publishing Cor.

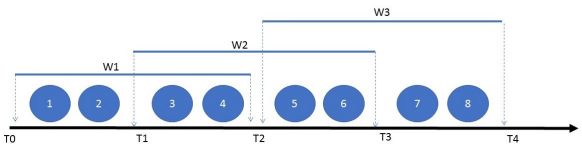

Fig. 2: Sliding Window.

Algorithm 2 Parallel Window Based Local Outlier Detection (PWLOD)

input: : BBGP virtual boarder percentage *vbp*, inter-window percentage *iwp*, LOF threshold ε, processing window size *pws*, processing node catalog *pnc*, number of grids in each dimension *gd*, dataset *D*
output: Outliers
1. Initialize waiting queue *wq* = {}
2. Initialize processing queue *pq* = {}
3. **For** every tuple t arrive in the data stream
4. **If** the number of records in waiting queue less than *pws* **then**
5. Add *t* to *wq*
6. **Else**
7. Add the top *pws* records from *wq* to the *pq*
8. ***NG*** Nodes allocated with grids Calling Algortihm 1 with *pnc* , *gd* , *pq* , *vbp*
9. **For** every node in *NG* decodes and send the grid to the specified processing node
10. **For** every processing node finishes its processing merge its grids
11. **For** records *tTemp* in every grid
12. **If** the LOF for *tTemp* is greater than ε **then**
13. *tTemp* is outlier
14. Empty the *pq*
15. Remove the top (*pws* * *iwp*) records from *wq*
16. **End**

4 Results and Discussion

In this section we explain the experimental results of the PWLOD algorithm and comparison the results with DILOF algorithm in [7]. We use docker [27] as Container Platforms. A docker-compose script was written to configure five services, one volume and a main network between them. The five services are one for the master node and four for the processing nodes. The data was saved on the volume shared between them. The 5 services and the volume have connection between each other through the main network. The DILOF implemented in C++ and the source code for it is available at (http://di.postech.ac.kr/DILOF). Both of the algorithms are deployed over a machine has a 2.5 GHz Intel Core i7 CPU, 16G memory DDR3, and 512G SSD hard disk.

We use Area Under the ROC Curve (AUC) criteria and the execution times to evaluate the PWLOD algorithm on a set of real-world data obtained from the Machine Learning database repository at UCI [28]. The datasets are:

1. UCI Vowel: consists of 1,456 data points, 12 dimension and 11 classes.
2. KDD Cup 99: consists of 95,156 data points and 3 dimensions.
3. Covertype: consists of 286,048 data points and 10 dimensions.

For the experimental setup The hyper-parameters of DILOF, η and λ are fixed to 0.3 and 0.001 for all datasets. And the number of iterations is set to be 20. Also, we set Border percentage *B* for the BBGP algorithm to be 25%. Also, we set *k* to be 19 for the UCI Vowel dataset and 8 for the both KDD Cup 99 smtp and covertype datasets. For the summarization phase in DILOF we set the window limit *w* to be *w* = 200, 300, 400, 500 for the PWLOD algorithm we set processing window size *pws* = 200, 300, 400, 500 for all the datasets. For inter-window percentage *iwp* in the PWLOD we set them to be 25% and 50% of *pws*.

4.1 The outlier detection accuracy

We evaluate the AUC for the two algorithms PWLOD and DILOF. Figs. 3–5 show the AUC for the three datasets UCI Vowel, KDD Cup 99 smtp and covertype respectively. For all the datasets the we notice that the AUC value for the DILOF algorithm is higher than the AUC value for the 25% PWLOD algorithm in the small size of window. But when the window size increases the AUC value for the DILOF increases but with a little value in contrast the AUC value for the PWLOD increases with larger value than the AUC value for DILOF. The AUC values increase till the difference between the two AUC values for the DILOF and 25% PWLOD algorithm reaches near zero in window size 500.

The same is for the 50% PWLOD the AUC value for the DILOF starts with higher value than the 50% PWLOD in window size 200 and increases till the AUC value for the PWLOD reaches higher value than the AUC value for DILOF in window size 500. We conclude that the increase in the size of the window affects the AUC value for the 25% and 50% PWLOD higher than the AUC value for DILOF. This is due to that when we increase the window size the number of processed points become large which increases the detection accuracy for the outliers. We notice that for all the datasets the difference between the AUC for the two algorithms starts to be big and become smaller as the window size increases but don't reach zero or negative value as the AUC for the PWLOD are always higher than the AUC of DILOF.

4.2 Execution Time

Figs. 6–8 show the AUC for the three datasets UCI Vowel, KDD Cup 99 smtp and Covertype respectively. As shown the execution time for the 25% and 50% PWLOD

© 2020 NSP
Natural Sciences Publishing Cor.

Inf. Sci. Lett. **9**, No. 3, 211-217 (2020) / www.naturalspublishing.com/Journals.asp

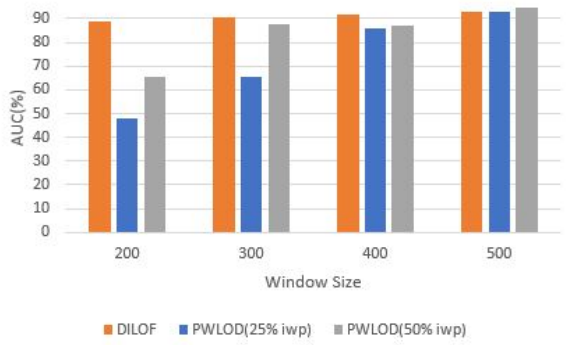

Fig. 3: UCI Vowel Dataset Outlier Detection Accuracy using AUC.

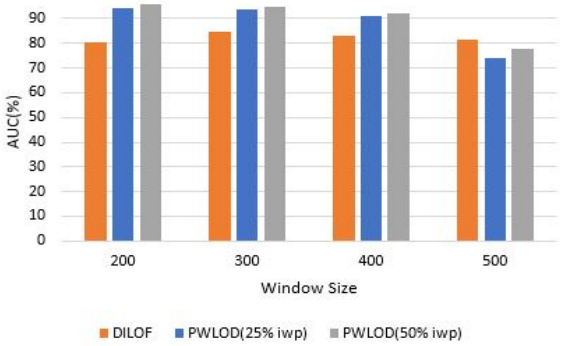

Fig. 4: KDD Cup 99 smtp Dataset Outlier Detection Accuracy using AUC.

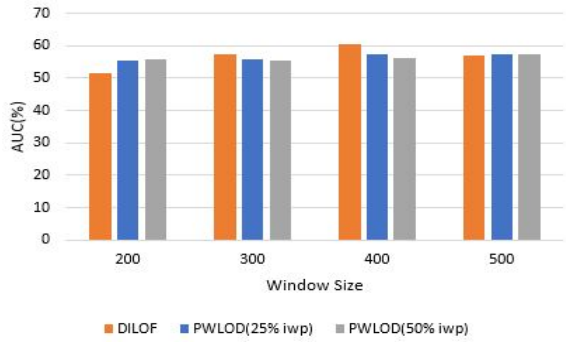

Fig. 5: Covertype Dataset Outlier Detection Accuracy using AUC.

algorithm is always much lower than the DILOF algorithm. This is because the PWLOD algorithm distribute the load between the processing node and not only any distribute but also balanced distribution. The main drawback is that there is some redundancy between nodes in the points located in the edges of the grids.

Also, the processing time for the 25% is lower than the 50% as the size of the data that each node processes is smaller and so the distance matrix for the LOF algorithm is smaller $O(n^2)$. Also, we notice that when the window size increases the DILOF algorithm execution time increases but in contrast for both 25% and 50% PWLOD algorithm the execution time remains almost the same but with little increase. For the covertype dataset the execution time for the 50% PWLOD algorithm is about 10x smaller than the time for DILOF.

The execution time for the 25% and 50% PWLOD algorithm is 10x faster than the execution time for the DILOF and the AUC value for both of them are almost the same.

Finally, the DILOF algorithm may have small higher value for the AUC in some window sizes than the 25% and 50% PWLOD but the 25% and 50% PWLOD execution time is 10x faster than the execution time for the DILOF algorithm.

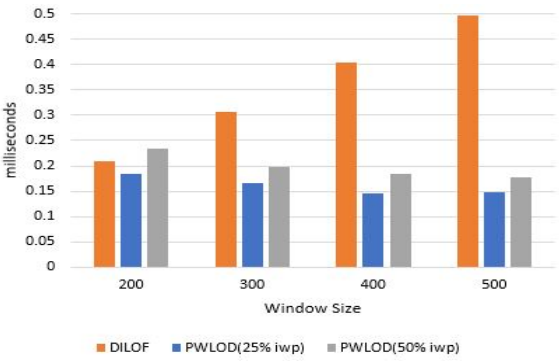

Fig. 6: UCI Vowel Dataset Execution Time.

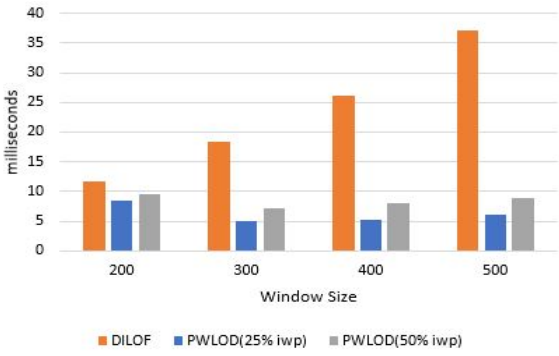

Fig. 7: KDD Cup 99 smtp Dataset Execution Time.

© 2020 NSP
Natural Sciences Publishing Cor.

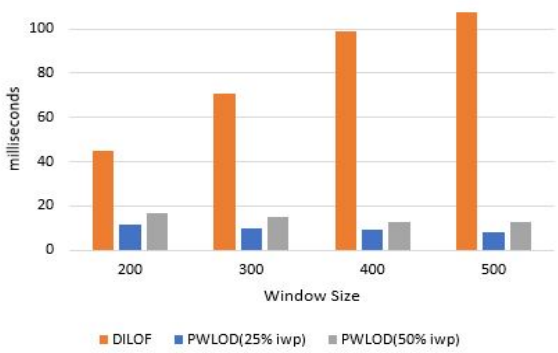

Fig. 8: Covertype Dataset Execution Time.

5 Conclusion

LOF is one of the algorithms that detects outliers in static data but it has limitations when dealing with data streams. 1) it consumes a lot of memory as the whole data need to be stored in the memory (which isn't applicable in data stream as the data size is infinite). 2) it needs to process the whole data once and any change in the data require that the LOF to be recalculated from the beginning (which isn't applicable in data stream as the data is changing). We propose a novel algorithm called PWLOD which overcome the two limitation of the LOF in data stream and detect outliers in distributed environment processed in parallel. Our experimental evaluations demonstrate that PWLOD has 10x faster execution time than the state-of-the-art competitors in detecting the outliers and has almost the same Accuracy in the detection with state-of-the-art competitors.

Conflict of Interest

The authors declare that there is no conflict of interest regarding the publication of this article.

References

[1] S. Sadik and L. Gruenwald, "Research issues in outlier detection for data streams," ACM SIGKDD Explorations Newsletter, vol. 15, pp. 33-40, 2014.

[2] M. Salehi, C. Leckie, J. C. Bezdek, T. Vaithianathan and X. Zhang, "Fast memory efficient local outlier detection in data streams," IEEE Transactions on Knowledge and Data Engineering, vol. 28, pp. 3246-3260, 2016.

[3] Y. Yan, L. Cao, C. Kulhman and E. Rundensteiner, "Distributed local outlier detection in big data," in Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017.

[4] Y. Yan, L. Cao and E. A. Rundensteiner, "Scalable Top-n Local Outlier Detection," in Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017.

[5] A. K. Jain, "Data clustering: 50 years beyond K-means," Pattern recognition letters, vol. 31, pp. 651-666, 2010.

[6] M. Sakr, W. Atwa and A. Keshk, "Distributed Anomaly Detection Over Big Data," Research Journal of Applied Sciences, Engineering and Technology, vol. 16, pp. 77-87, 2019.

[7] G. S. Na, D. Kim and H. Yu, "DILOF: Effective and Memory Efficient Local Outlier Detection in Data Streams," in Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2018.

[8] M. Sakr, W. Atwa, and A. Keshk. "Sub-Grid Partitioning Algorithm for Distributed Outlier Detection on Big Data." In 2018 13th International Conference on Computer Engineering and Systems (ICCES), pp. 252-257. IEEE, 2018

[9] V. Chandola, A. Banerjee and V. Kumar, "Anomaly detection: A survey," ACM computing surveys (CSUR), vol. 41, p. 15, 2009.

[10] D. Pokrajac, A. Lazarevic and L. J. Latecki, "Incremental local outlier detection for data streams," in Computational intelligence and data mining, 2007. CIDM 2007. IEEE symposium on, 2007.

[11] W. Atwa, and L. Kan. "Semi-supervised Clustering Method for Multi-density Data." In International Conference on Database Systems for Advanced Applications, pp. 313-319. Springer, Cham, 2015

[12] C. C. Aggarwal, "Outlier analysis," in Data mining, 2015.

[13] E. M. Knox and R. T. Ng, "Algorithms for mining distancebased outliers in large datasets," in Proceedings of the international conference on very large data bases, 1998.

[14] F. Angiulli and F. Fassetti, "Detecting distance-based outliers in streams of data," in Proceedings of the sixteenth ACM conference on Conference on information and knowledge management, 2007.

[15] D. Yang, E. A. Rundensteiner and M. O. Ward, "Neighbor-based pattern detection for windows over streaming data," in Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, 2009.

[16] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas and Y. Manolopoulos, "Continuous monitoring of distance-based outliers over data streams," in Data Engineering (ICDE), 2011 IEEE 27th International Conference on, 2011.

[17] M. M. Breunig, H.-P. Kriegel, R. T. Ng and J. Sander, "LOF: identifying density-based local outliers," in ACM sigmod record, 2000.

[18] K. Yamanishi, J.-I. Takeuchi, G. Williams and P. Milne, "On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms," Data Mining and Knowledge Discovery, vol. 8, pp. 275-300, 2004.

[19] K. Yamanishi and J.-i. Takeuchi, "A unifying framework for detecting outliers and change points from non-stationary time series data," in Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, 2002.

[20] W. Atwa, and L. Kan. "Clustering evolving data stream with affinity propagation algorithm." In International Conference on Database and Expert Systems Applications, pp. 446-453. Springer, Cham, 2014

© 2020 NSP
Natural Sciences Publishing Cor.

[21] W. Atwa, and L. Kan. ”Constraint-based clustering algorithm for multi-density data and arbitrary shapes.” In Industrial Conference on Data Mining, pp. 78-92. Springer, Cham, 2017.
[22] W. Atwa, and L. Kan. ”Affinity propagation-based clustering for data streams.” Applied Mathematics & Information Sciences 9, no. 4, 2015.
[23] C. C. Aggarwal, J. Han, J. Wang and P. S. Yu, ”A framework for projected clustering of high dimensional data streams,” in Proceedings of the Thirtieth international conference on Very large data bases-Volume 30, 2004.
[24] C. C. Aggarwal, ”A segment-based framework for modeling and mining data streams,” Knowledge and information systems, vol. 30, pp. 1-29, 2012.
[25] I. Assent, P. Kranen, C. Baldauf and T. Seidl, ”Anyout: Anytime outlier detection on streaming data,” in International Conference on Database Systems for Advanced Applications, 2012.
[26] M. Salehi, C. A. Leckie, M. Moshtaghi and T. Vaithianathan, ”A relevance weighted ensemble model for anomaly detection in switching data streams,” in Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2014.
[27] esotericsoftware, ”kryonet”.
[28] D. Dheeru and E. Karra Taniskidou, UCI Machine Learning Repository, 2017.

© 2020 NSP
Natural Sciences Publishing Cor.