Journal of Computing and Security

Research Article, 2023, Volume 10, Number 1 (pp. 35–46)

http://www.jcomsec.org

Summarization Algorithm for Data Stream to Speed up Outlier Data Detection

Hadid Mollashahi [a], Hamid Saadatfar [a], Hamed Vahdat-Nejad [a,*]

[a] *Faculty of Electrical & Computer Engineering, University of Birjand, Birjand, Iran.*

ARTICLE INFO.

Article history:
Received: 7 September 2022
Revised: 17 February 2023
Accepted: 15 March 2023
Published Online: 30 May 2023

Keywords:
Outlier Detection, Data Stream, Machine Learning, Clustering, IoT.

ABSTRACT

Outlier detection in data streams is an essential issue in data processing. Today, due to the massive growth of streaming data generated by the spread of the Internet of Things, outlier detection has become a significant challenge. Much progress has been made in outlier detection based on local outlier detection algorithms, such as density-based local outlier factor algorithms, suitable for static data. The incremental version of these algorithms is used to detect the local outliers in streaming data. However, outlier detection in streaming data faces the challenges of limited memory capacity, high execution time, inaccessibility of all data at one time, and changes in data distribution (increasing and decreasing input rates, uncertainty, etc.). In this paper, we propose a density-based summarization algorithm, which summarizes data, every time the buffer is filled. The proposed algorithm maintains the desired shape of the clusters, with a low computational cost. To this end, larger clusters are selected and the data of their dense areas are reduced so that the shape of the old clusters is not lost. The proposed summarization algorithm reduces execution time and increases precision, recall, and F1 score compared with the evaluated algorithms.

1 Introduction

Data mining is the process of analyzing large volumes of data used in many areas of research, including sales and marketing, product development, healthcare, and education. The process fails if the samples do not indicate a good representation of the system under learning [1, 2]. Practical data mining operations require reasonable and appropriate data. Data objects that do not conform to the general behavior of the system or trends underlying the generation of the dataset are called outliers [3]. Outlier detection seeks to find patterns in the data that do not conform to the expected behavior [3]. Outlier detection has many applications, including in the field of medicine and public health [4], credit card fraud detection [5], environmental monitoring and network intrusion detection, etc. [6, 7], some of which are general and others specific.

Generally, outlier detection is done with three different approaches, which are supervised, unsupervised, and semi-supervised methods [8, 9]:

- In the supervised method, the data is labeled either normal or an outlier. The challenge with this method is that the classes are unbalanced, meaning that the outlier data population is typically

* Corresponding author.
Email addresses: hdmollashahi@birjand.ac.ir (H. Mollashahi), saadatfar@birjand.ac.ir (H. Saadatfar), vahdatnejad@birjand.ac.ir (H. Vahdat-Nejad)
https://dx.doi.org/10.22108/JCS.2023.135054.1109
ISSN: 2322-4460

smaller than the normal sample.
- In the unsupervised method, the data is not labeled normal/outlier. It is assumed that normal data follows patterns more frequently than outliers. The challenge is that sometimes normal objects may not share a strong pattern, and collective junk data may leave a strong pattern in small environments, making it difficult to distinguish.
- In the semi-supervised method, data samples have only the normal label and do not need the outlier label [10, 11].

The Local Outlier Factor (LOF) [12] is a well-known density-based algorithm to find local outliers in static data. LOF measures data points' local deviations for their K-nearest neighbors, where K is a user-defined parameter. Numerous applications, including fraud and intrusion detection, can benefit from this technique. Later, incremental databases and stream environments were added to the LOF paradigm [13]. In this regard, outlier detection algorithms based on LOF have been developed for data streams, such as MILOF, DILOF, and TADILOF [14–16]. Data stream algorithms should prevent the use of obsolete data. By summing up earlier data points, data stream techniques use a predetermined window size to restrict the number of data points retained in memory.

This paper deals with the problem of outlier detection through unsupervised data streams. Since in data streams, all samples cannot be stored in the buffer, stream data processing methods summarize the data periodically. In this paper, a new idea for data summarization is presented. The proposed method summarizes data in each cluster and creates free space for new data with low computational costs. Since the proposed method preserves the shape of the clusters and removes the data based on the density, the obtained clusters should have higher accuracy and quality. Better detection of clusters helps to increase the accuracy of the detection of outlier data.

Following is the rest of this paper. In Section 2, we go over the relevant research. The recommended approach is then presented in Section 3. Our tests and an analysis of the suggested method's performance are described in Section 4. Section 5 presents the conclusion of the essay.

2 Related Works

Outlier detection is getting significant attention in the research field of big data. Detecting the outlier is important in various applications such as communication, finance, fraud detection, and network intrusion detection. Data streams posed new challenges to the existing algorithms of outlier detection. Local Outlier Factor (LOF) [12] is one of the most appropriate techniques used in the density-based method to determine the outlier. However, it faces some difficulties regarding data streams. First, LOF processes the data all at once, which is not suitable for data streams. Another issue appears when a new data point arrives. All the data points need to be recalculated again significantly.

The ILOF algorithm also dynamically adjusts the profiles of data points. Due to the possibility of changing data profiles over time, this capability is essential. Evidence proves that only a small portion of a data point's close neighbors are impacted by either adding a new data point or deleting an existing one. As a result, the total number of points in the set has no bearing on the number of updates per insertion or deletion. Every data record introduced into the data set has its LOF value calculated by the ILOF algorithm, which then assesses whether it is an outlier. Additionally, LOF values are changed for current data records as necessary. MILOF [14] has been developed to find outliers in data streams while utilizing little memory. It solves the memory issue by summing up data points. The three stages of MILOF implementation are insertion, summarization, and integration. The similarity between the MILOF insertion and ILOF step should be noted. When the number of memory-stored points exceeds the window size b limit, the summing process is initiated. After locating the centers of the c cluster, which will serve as the initial $b/2$ of the data, the insertion phase is performed using the K-means algorithm. Each cluster center receives a weight in the integration stage based on the number of associated data points. The new and old cluster centers are then combined using the K-means weighting technique.

MILOF can be utilized to decrease memory and processing requirements. However, the structure and shape of the clusters in the original data are not preserved in summarization by this method, which is crucial for precise detection. DILOF, a local technique for outlier detection based on density for data streams, is similar to MILOF in that it employs the LOF value to identify outliers. The two stages of DILOF are identification and summarization [15]. When additional data points are introduced to the dataset, the detection phase calculates LOF values using the ILOF algorithm. The data points are then categorized by DILOF into a regular class or outlier. When the number of data points approaches the window size limit, the summarization phase, known as the non-parametric density summary, is enabled for the summary candidate. The optimum summary composition is then chosen using the descending slope approach. By minimizing the loss function, the summary can capture 50 percent of the data ($W/2$) in a quarter of the window size.

In an unlimited stream data environment [16], the status of the current data point must be determined as outlier / internal data before moving on to the following data points. Remember that the following data points in the data stream are not mentioned. Additional clusters could be produced by adding new data. Before new data join the data stream, a restricted amount of computational power must be employed. Because of this, algorithms need to be quick to execute. TADILOF [16] calculates the estimated LOF score using summaries of prior data points. Assessing the distance between recently added data points and probably deleted neighbors falls under this category. For instance, in the preceding summary stage, data points were removed. This method uses the LOF score to determine if a newly added data point satisfies the LOF threshold. A secondary examination depending on the estimated LOF score proposal is performed. It determines whether a newly inserted data point is a throwaway if the LOF threshold indicates that it is. The data was kept in the window using the idea of a static window strategy.

A new algorithm is proposed in [17] called Grid Partition-based Local Outlier Factor (GP-LOF). GP-LOF uses a grid for the LOF with a sliding window to detect outliers.

In ELOF [18], a memory window mechanism was designed to limit the amount of data storage. Then, a new sub-data mining model was designed to extract the sub-data of the original data information. Through these two plans, the amount of data storage can be effectively reduced. As mentioned, most outlier detection papers focus on reducing less important data. The proposed summarization algorithm reduces execution time and increases precision, recall, and F1 score compared with the evaluated algorithms.

3 Proposed Algorithm

To detect outliers in streaming data, we face two challenges: the impossibility of real-time detection and limited memory. The memory issue is an important requirement in the field of outlier detection for data streams. In this research, to deal with the challenge of identifying outliers in real time, the incremental LOF algorithm is used. ILOF requires storing all previous data points to calculate local outliers. This problem is overcome by purposefully summarizing the stream data samples. In the proposed summarization method, the clustered samples that belong to dense areas have more chances to be removed. In this way, newly formed clusters and boundary data are more supported. Better detection of clusters and not losing the shape of clusters help to identify outliers with better accuracy.

The proposed algorithm consists of two phases. The first phase is the detection stage. In this phase, we use the ILOF algorithm [13] to detect whether a point is distant or internal. The streaming data enters the window upon arrival. The ILOF algorithm determines whether the new sample is an outlier or an internal sample. Outlier data points are added to a set called the outlier data set. Then the historical LOF and finally the approximate LOF algorithm [16] are used to detect the final state of the current point. The second phase involves summarizing the data in memory (buffer).

This phase attempts to increase the free space for the entry of new data, manage changes in data distribution (increase and decrease in entry rates, uncertainties, etc.) that may change over time, and ultimately synchronize the speed of information processing with the speed of data arrival.

The proposed algorithm allocates part of the memory (window) for the summarized older data and the other part for storing new data. In the detection phase, the LOF value is calculated for each input point; the items required to calculate the approximate LOF are updated according to the data summarized in the previous steps, such as LRD, LOF, and mean distance, which are addressed in the TADILOF algorithm [16]. The proposed Summarization Algorithm (SALG) is called when the window is filled with data points. SALG summarizes half of the data points in the window and erases them from memory. The pseudocode of SALG is as follows:

Where the delete function is as follows:

The first to third lines of the SALG algorithm calculate the LOF value for each input data point. When the LOF value of the entry point is less than the historical LOF, the items needed to calculate the approximate LOF are updated according to the data summarized in the previous steps (LRD, LOF, and mean distance). In lines 6 to 10 of SALG, when the window (b_w) is filled with data points, the summarization algorithm is called. By summarizing and deleting data points to the desired limit, $b_w/2$ data points are cleared from memory. In the proposed summarization algorithm, we consider a quota for each cluster. Assuming (a) the selection of $b_w/2$ data from the window to summarize and delete, and (b) the number of clusters is n, the final contribution of each cluster is $b_w/2n$. After the dense clusters are identified, the larger clusters, which contain the largest shares of the data, are selected for summarization and deletion, because the goal is to make dense clusters smaller.

In line 6, the summarization is initiated. Whenever the buffer capacity is full, the summarization process

Algorithm 1 Summarization Algorithm (SALG)

Input: :Set of data points in memory $X = x_1, x_2, \ldots, x_w$
Windows $size = b_w$
Memory size limit for $delete = (b_w)/2$
Number of data points in cluster $i = |c_i|$
$Numberofclustersinmemory = n$
$Numberofneighbors = k$

Output: : Summary set
for each data point $x \in X$ **do**
if $lof_k(x) <$ historial lof(x) **then**
Update lof,lrd and mean distance
end if
end for
$S = \{\}$
if number of data points in $memory = b_w$ **then**
$Num = 0$
end if
for all $c_i \in C$ **do**
if $\|c_i\| \leq (b_w/2n + Num)$ **then**
$num = num + (b_w/2n - |c_i|)$
else
queue $Q \leftarrow$ Sorting elements of c_i in an increasing order by ApproximateLOF formula in eq (5-3)
$c_i \leftarrow Delete(Q, b_w/2n + num)$
end if
$S \leftarrow S \cup c_i$
end for
Return s

Algorithm 2 Delete Function

function DELETE$(Q, size)$
$C = \{\}$
do
Add First element of Q to C
Delete k nearest neighbors of the first element from Q
while $|Q| > Size$
$C \leftarrow C \cup Q$
Return C
end function

will be executed. Half of the data in the buffer $(b_w/2)$ should be deleted, and free space for new data should be created. In this regard, an equal share of the buffer space is considered for the current clusters. The algorithm starts removing samples from clusters that have more data. The proposed method tries to create a balance between the clusters in the process of removing data and does not remove data from emerging and small clusters (line 9). As a result, the proposed method starts to remove data from clusters that have more samples than other clusters. In these clusters,

Table 1. Dataset

Dataset	#Data Points	#Dimensions	#Outlier Data Points	Need to shuffle
Annthyroidnt	7200	6	534	False
Cardio	1831	21	176	True
HTTP (KDD Cup 99)	567,498	3	2211	False
Letter Recognition	1600	32	100	True
Mnist	7603	100	700	True
Musk	3062	166	97	True
Pendigits	6870	16	156	False
Satellite	6435	36	2036	False
SMTP (KDD Cup 99)	95156	3	30	False
Vowels	1456	12	50	True

the data are sorted in ascending order based on the LOF index that was previously calculated for all samples. The lower value of the LOF index for a sample means that we have a higher density around that sample. Therefore, we will have a queue of data of a cluster. The first sample in this queue will have the highest density. The process of deleting data in this cluster starts by checking samples from the front of this queue. The densest data remains in the buffer as a representative, and its neighbors are removed from the queue. The same process is executed for the next instance in the queue to remove the required amount of data (according to the cluster's share of the buffer space) from this cluster. In this way, the data that are in the border areas and have a lower density remain in a cluster. This helps the cluster to lose less of its overall structure and shape. Removing data from large clusters based on density and keeping the dense neighbor of that data as a representative allow both the shape of old clusters to be preserved and emerging clusters to grow and not be mistakenly considered as noise. It should be noted that to find dense data samples, new calculations are not imposed on the method, but this work is done based on the LOF index, which is calculated for all input data.

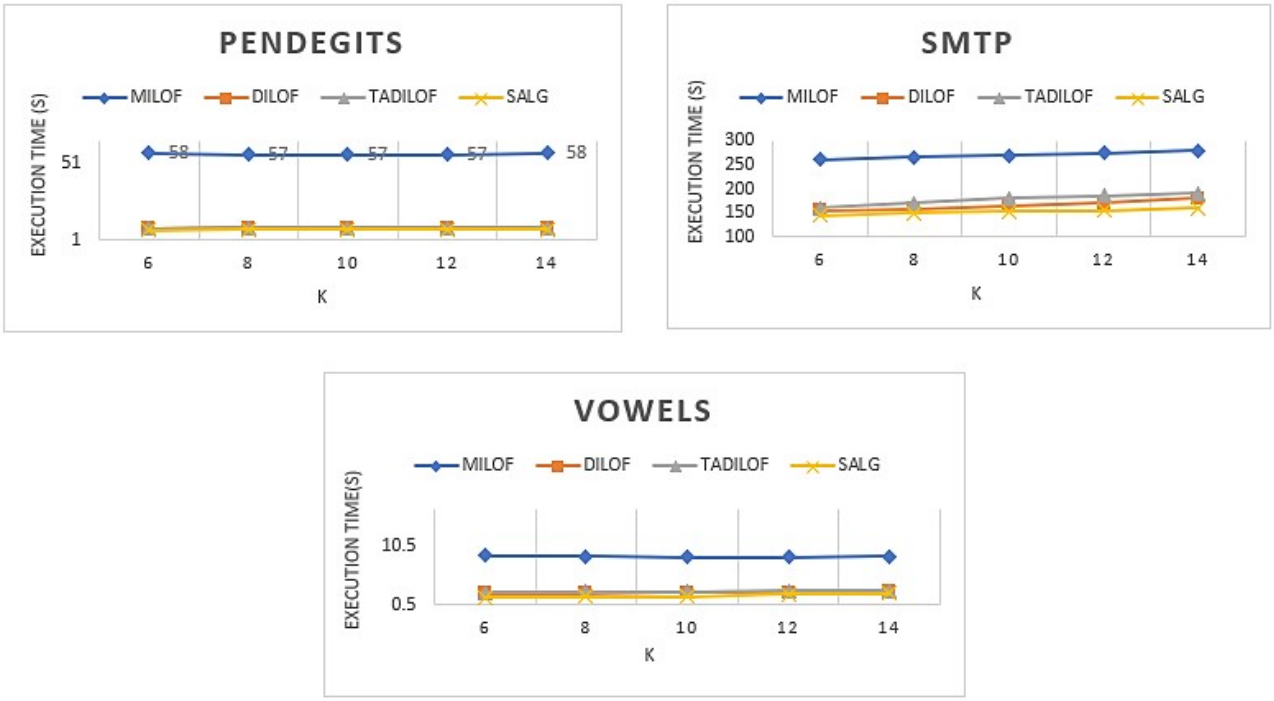


Figure 1. Execution time in Pendigits, SMTP, and Vowels datasets.

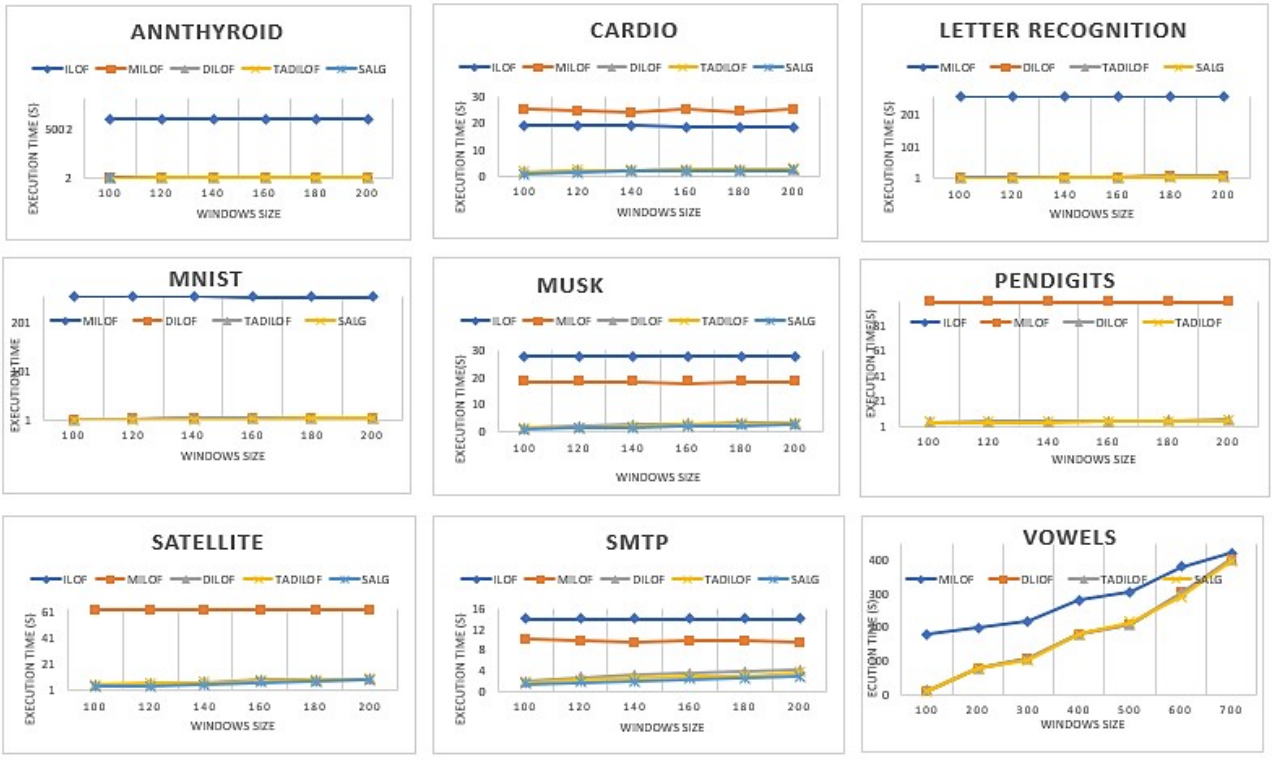


Figure 2. Execution time on 9 data sets with different window sizes and K = 8.

4 Experimental Evaluation

This section evaluates the proposed SALG algorithm and compares it with other recent related ones. The performance is evaluated on different data sets, shown in Table 1. These preprocessed datasets (Datasets) have been widely used in previous works to evaluate the methods. They belong to the Machine Learn-

Table 2. Precision, Recall, F1 Score in Annthyroid Dataset.

	Precision				Recall				F1 Score			
Window Size	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG
100	0.2591	0.2242	0.2290	0.2344	0.3502	0.3839	0.3507	0.4008	0.2978	0.2831	0.2770	0.2958
120	0.2648	0.2227	0.2331	0.2301	0.356	0.3927	0.3582	0.4064	0.3037	0.2843	0.2825	0.2938
140	0.2599	0.2138	0.237	0.2176	0.3678	0.4043	0.3631	0.4082	0.3046	0.2796	0.2868	0.2839
160	0.2575	0.2186	0.2382	0.2258	0.3787	0.4157	0.3680	0.4288	0.3065	0.2865	0.2892	0.2959
180	0.2588	0.2175	0.2464	0.2185	0.3751	0.4184	0.3727	0.4195	0.3063	0.2862	0.2967	0.2873
200	0.2646	0.2184	0.2434	0.2231	0.3809	0.4268	0.3751	0.4345	0.3123	0.2890	0.2952	0.2948

Table 3. Precision, Recall, F1 Score in Cardio Dataset.

	Precision				Recall				F1 Score			
Window Size	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG
100	0.3467	0.3694	0.3152	0.4009	0.3914	0.4548	0.3051	0.5057	0.3677	0.4076	0.3100	0.4472
120	0.3381	0.3508	0.3285	0.3909	0.3751	0.4397	0.3127	0.4886	0.3557	0.3903	0.3204	0.4343
140	0.3218	0.3431	0.3028	0.3839	0.3656	0.4324	0.2983	0.4886	0.3423	0.3826	0.3005	0.4300
160	0.3152	0.3355	0.3063	0.3474	0.3570	0.4157	0.3038	0.4205	0.3348	0.3713	0.3050	0.3805
180	0.3210	0.3262	0.3022	0.3593	0.3513	0.4148	0.3022	0.4716	0.3354	0.3652	0.3022	0.4078
200	0.3121	0.3206	0.2920	0.3482	0.3422	0.4043	0.2994	0.4432	0.3265	0.3576	0.2957	0.3900

ing Database Library of the University of California, Irvine[19].

calculated for all samples. The lower value of the LOF index for a sample means that we have a higher density around that sample. Therefore, we will have a queue of data of a cluster. The first sample in this queue will have the highest density. The process of deleting data in this cluster starts by checking samples

from the front of this queue. The densest data remains in the buffer as a representative, and its neighbors are removed from the queue. The same process is executed for the next instance in the queue to remove the required amount of data (according to the cluster's share of the buffer space) from this cluster. In this way, the data that are in the border areas and have a lower density remain in a cluster. This helps the cluster to lose less of its overall structure and shape. Removing data from large clusters based on density and keeping the dense neighbor of that data as a representative allow both the shape of old clusters to be preserved and emerging clusters to grow and not be mistakenly considered as noise. It should be noted that to find dense data samples, new calculations are not imposed on the method, but this work is done based on the LOF index, which is calculated for all input data.

The results of the MILOF method are taken from reference articles [16]. We only calculated the results of our method (SALG) and finally compared them with the results of previous papers. We compared the performance of SALG with state-of-the-art algorithms, DILOF [15] and MILOF [14]. We downloaded DILOF and ILOF implementations from the URL provided in [15]. The conditions in the implementation order to make a run of all methods have been the same in a fair comparison. Some of the preprocessed datasets contained all the outliers grouped (as a class) at the beginning or end. Some datasets had outliers scattered among inliers. Therefore, in TADILOF [16], before running the algorithms, the data set of the previous type was shuffled. The value of the K parameter for finding the nearest neighbors is assumed to be 8 for all datasets, such as recommended in DILOF [15]. Because of the significance of window size in terms of memory usage and calculation time of the algorithm, the models have also been examined using windows of various widths. A small window with the sizes W = {100, 120, 140, 160, 180, 200} is chosen for small data sets. Window sizes W = {100, 200, 300, 400, 500, 600, 700} are taken into consideration for larger datasets. For the LOF score threshold, the same thresholds are used in the evaluation, which include these values: {0.1, 1.0, 1.1, 1.15, 1.2, 1.3, 1.4, 1.6, 2.0, 3.0}. All algorithms were implemented in MATLAB programming language.

To carry out a comprehensive evaluation, we examine the execution time (speed), Precision, Recall, and F1 Score for various window sizes and K = 8. In most cases, SALG has better execution time and better precision, recall, and F1 score than similar recent related algorithms. The algorithms are executed ten times on each data set. The presented results are the average of these ten runs.

The runtime was first computed for the Pendigits, SMTP, and Vowels datasets (Figure 1). The window size for the Pendigits and Vowels datasets is set to 140. At the same time, the window size is set to 400 for the SMTP dataset. Parameter K is displayed in different sizes 6, 8, 10, 12, and 14 (horizontal axis). With varying sizes of the K parameter, the execution time of the SALG algorithm is much less than MILOF and has improved compared to other algorithms. The results of Figure 1 show that the proposed method performs better than the competitors for different values of the K parameter. It also shows that the performance of the proposed method is not very sensitive to the values of this parameter.

Figure 2 shows the execution time of algorithms on different datasets with different window sizes and K = 8. The vertical axis is the runtime (in seconds), and the horizontal axis is the size of the window. In the Annthyroid, Mnist, Musk, Pendigits, and Satellite datasets, execution time is at the base two logarithmic scale. All three DILOF, TADILOF, and SALG algorithms are significantly better than ILOF and MILOF in runtime. SALG execution time is less than other algorithms.

The precision, recall, and F1 score criteria for different window sizes and K = 8 have been investigated in the studied data sets. Tables 2 to 9 show the precision, recall, and F1 score in the various datasets for DILOF, TADILOF, MILOF, and SALG. The SALG algorithm achieves better precision, recall, and F1 score in most cases.

Table 3 shows the precision, recall, and F1 score in the Cardio dataset which has 1831 data points, 21 dimensions, and 176 outliers for DILOF, TADILOF, MILOF, and SALG algorithms. In most cases, the SALG algorithm achieved better precision, recall, and F1 score.

Table 4 shows the accuracy, recall, and F1 score in the Letter Recognition dataset which has 1600 data points, 32 dimensions, and 100 outliers for DILOF, TADILOF, MILOF, and SALG algorithms. In most cases, the SALG algorithm achieved better precision, recall, and F1 score.

Table 5 shows the precision, recall, and F1 score on the Mnist dataset which has 7603 data points, 100 dimensions, and 700 outliers for DILOF, TADILOF, MILOF, and SALG algorithms. In most cases, the SALG algorithm achieved better precision, recall, and F1 score.

Table 6 shows the precision, recall, and F1 score in the Musk dataset which has 3062 data points, 166 dimensions, and 97 outliers for DILOF, TADILOF, MILOF, and SALG algorithms. In most cases, the

Table 4. Precision, Recall, F1 Score in Letter Recognition Dataset.

	Precision				Recall				F1 Score			
Window Size	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG
100	0.1270	0.1124	0.2220	0.1386	0.2059	0.2308	0.2593	0.2800	0.1571	0.1512	0.2392	0.1854
120	0.1482	0.1632	0.2437	0.1953	0.2139	0.2443	0.2616	0.2500	0.1751	0.1957	0.2523	0.2193
140	0.1547	0.1557	0.2299	0.1677	0.2193	0.2517	0.2618	0.2600	0.1814	0.1924	0.2448	0.2039
160	0.1711	0.1738	0.2575	0.2368	0.2271	0.2625	0.2663	0.3600	0.1951	0.2091	0.2618	0.2857
180	0.1977	0.2015	0.2839	0.2248	0.2335	0.2706	0.2718	0.2900	0.2141	0.2310	0.2777	0.2533
200	0.2008	0.1985	0.2843	0.2205	0.2375	0.2732	0.2696	0.2800	0.2176	0.2300	0.2768	0.2467

Table 5. Precision, Recall, F1 Score in Mnist dataset.

	Precision				Recall				F1 Score			
Window Size	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG
100	0.2214	0.1916	0.2404	0.2140	0.2091	0.2437	0.2414	0.2714	0.2150	0.2145	0.2409	0.2393
120	0.2003	0.2726	0.2541	0.2973	0.2122	0.2483	0.2423	0.2629	0.2061	0.2599	0.2481	0.2790
140	0.1987	0.2605	0.2927	0.2797	0.2164	0.2571	0.2491	0.2757	0.2072	0.2587	0.2691	0.2777
160	0.2285	0.2939	0.3016	0.3296	0.2172	0.2621	0.2524	0.2914	0.2227	0.2771	0.2748	0.3093
180	0.1773	0.2813	0.3080	0.3232	0.2194	0.2659	0.2572	0.3043	0.1961	0.2734	0.2803	0.3135
200	0.1866	0.2970	0.2982	0.3260	0.2219	0.2706	0.2574	0.2971	0.2027	0.2832	0.2763	0.3109

SALG algorithm achieved better precision, recall, and F1 score.

Table 7 shows the accuracy, recall, and F1 score in the Pendigits dataset which has 6870 data points, 16 dimensions, and 156 outliers for DILOF, TADILOF, MILOF, and SALG algorithms. In most cases, the SALG algorithm achieved better precision, recall, and F1 score.

Table 6. Precision, Recall, F1 Score in Musk dataset.

	Precision				Recall				F1 Score			
Window Size	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG
100	0.4422	0.4141	0.4397	0.4400	0.3313	0.5407	0.2926	0.5670	0.3788	0.4690	0.3514	0.4955
120	0.4083	0.4028	0.4104	0.4643	0.2855	0.4830	0.2663	0.5361	0.3360	0.4393	0.3230	0.4976
140	0.3924	0.3882	0.4092	0.4455	0.2637	0.4541	0.2398	0.5052	0.3154	0.4186	0.3024	0.4734
160	0.3857	0.3907	0.3660	0.4412	0.2462	0.4087	0.2361	0.4639	0.3005	0.3995	0.2870	0.4523
180	0.4098	0.3820	0.3577	0.4184	0.2323	0.3760	0.2065	0.4227	0.2965	0.3790	0.2618	0.4205
200	0.3929	0.3710	0.3544	0.3977	0.2199	0.3364	0.1968	0.3608	0.2820	0.3529	0.2531	0.3784

Table 7. Precision, Recall, F1 Score in Pendigits Dataset.

	Precision				Recall				F1 Score			
Window Size	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG
100	0.0540	0.0955	0.1031	0.1044	0.3122	0.4455	0.3919	0.4744	0.0921	0.1573	0.1633	0.1711
120	0.0517	0.1143	0.0898	0.1285	0.3224	0.4833	0.3850	0.5321	0.0892	0.1849	0.1456	0.2070
140	0.0583	0.0869	0.0877	0.0927	0.3314	0.4814	0.3945	0.5064	0.0991	0.1472	0.1434	0.1568
160	0.0553	0.0676	0.0736	0.0770	0.3301	0.4859	0.3857	0.5449	0.0948	0.1187	0.1236	0.1349
180	0.0430	0.0735	0.0746	0.0806	0.3141	0.4782	0.3844	0.5256	0.0756	0.1274	0.1249	0.1398
200	0.0500	0.0743	0.0828	0.0837	0.3282	0.4846	0.3872	0.5385	0.0868	0.1288	0.1364	0.1448

Table 8 shows the accuracy, recall, and F1 score in the Satellite dataset which has 6435 data points, 36 dimensions, and 2036 outlier data for DILOF, TADILOF, MILOF, and SALG algorithms. In most cases, the SALG algorithm achieved better precision, recall, and F1 score.

Table 9 shows the accuracy, recall, and F1 score in the SMTP dataset which has 95156 data points, 3

Table 8. Precision, Recall, F1 Score in Satellite Dataset.

	Precision				Recall				F1 Score			
Window Size	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG
100	0.4862	0.4883	0.4720	0.5032	0.2569	0.3338	0.2466	0.3468	0.3362	0.3965	0.3240	0.4106
120	0.4982	0.4964	0.4637	0.5135	0.2571	0.3419	0.2488	0.3551	0.3392	0.4049	0.3239	0.4199
140	0.4810	0.4940	0.4695	0.5099	0.2608	0.3328	0.2602	0.3428	0.3382	0.3977	0.3348	0.4100
160	0.4981	0.4921	0.4886	0.5059	0.2670	0.3257	0.2683	0.3369	0.3476	0.3920	0.3464	0.4045
180	0.4988	0.5079	0.4879	0.5227	0.2783	0.3391	0.2792	0.3502	0.3573	0.4067	0.3552	0.4194
200	0.4918	0.5051	0.4673	0.5194	0.2893	0.3415	0.2788	0.3482	0.3643	0.4075	0.3493	0.4169

Table 9. Precision, Recall, F1 Score in SMTP Dataset.

	Precision				Recall				F1 Score			
Window Size	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG
100	0.0027	0.0016	0.0024	0.0019	0.7633	0.7400	0.5029	0.8333	0.0053	0.0033	0.0048	0.0037
200	0.0026	0.0025	0.0025	0.0026	0.7733	0.7900	0.5147	0.8333	0.0051	0.0049	0.0050	0.0052
300	0.0033	0.0034	0.0028	0.0037	0.7933	0.8133	0.6179	0.8667	0.0066	0.0069	0.0056	0.0075
400	0.0027	0.0024	0.0027	0.0026	0.7867	0.9133	0.6603	0.9667	0.0053	0.0048	0.0053	0.0051
500	0.0019	0.0031 0	.0022	0.0032	0.7467	0.9467	0.6417	0.9667	0.0038	0.0062	0.0044	0.0065
600	0.0020	0.0019	0.0018	0.0019	0.7700	0.9767	0.5839	0.9667	0.0041	0.0038	0.0036	0.0038
700	0.0017	0.0019	0.0018	0.0020	0.6767	0.9067	0.5933	0.9667	0.0053	0.0033	0.0048	0.0040

Table 10. Precision, Recall, F1 Score in Vowels Dataset.

	Precision				Recall				F1 Score			
Window Size	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG	DILOF	TADILOF	MILOF	SALG
100	0.1434	0.1571	0.1922	0.1927	0.3256	0.3898	0.4302	0.4200	0.1991	0.2239	0.2657	0.2642
120	0.1689	0.1552	0.1959	0.1832	0.3476	0.4350	0.4202	0.4800	0.2273	0.2288	0.2672	0.2652
140	0.1683	0.1644	0.2007	0.1846	0.3660	0.4604	0.4350	0.4800	0.2306	0.2423	0.2746	0.2667
160	0.1721	0.1959	0.2394	0.2458	0.3756	0.4758	0.4384	0.5800	0.2361	0.2775	0.3097	0.3452
180	0.1604	0.1741	0.2276	0.2015	0.3862	0.5022	0.4494	0.5400	0.2267	0.2586	0.3021	0.2935
200	0.1644	0.1809	0.2074	0.2188	0.3914	0.5000	0.4348	0.5600	0.2315	0.2657	0.2809	0.3146

dimensions, and 30 outlier data for DILOF, TADILOF, MILOF, and SALG algorithms. In most cases, the SALG algorithm achieved better precision, recall, and F1 score.

Table 10 shows the precision, recall, and F1 score in the Vowels dataset, which has 1456 data points, 12 dimensions, and 50 outliers for DILOF, TADILOF, MILOF, and SALG algorithms. In most cases, the SALG algorithm achieved better precision, recall, and F1 score. The evaluation results show that the algorithm performs much better in large windows. It should be said that by increasing the size of the buffer, the flexibility and intelligence of the proposed method are shown more, and its performance improves more compared to previous similar methods.

5 Conclusions

In this research, a method for detecting outliers in the data stream has been presented, which uses a new density-based method (called SALG) to summarize the data inside the buffer. Using the LOF index, the proposed algorithm removes the data around dense samples. While maintaining the shape of micro-clusters, does not impose a lot of calculations on the algorithm. The SALG algorithm performed better in execution time, precision, recall, and F1 score than the previous algorithms. Regarding the weakness of the proposed method, it can be said that, like other similar methods, it requires determining the input value for its parameters, such as the size of the LOF neighborhood and the LOF threshold. Determining the appropriate value of how many neighboring samples of dense data can be removed can help the uniformity of density in the data as much as possible. The dynamic determination of this value can be considered for future research.

References

[1] CC. Aggarwal and PS. U. An effective and efficient algorithm for high-dimensional outlier detection. *The VLDB journal*, 14(2):211–21, 2005 Apr. doi:10.1007/s00778-004-0125-5.

[2] A. Arning, R. Agrawal, and P. Raghavan. A Linear Method for Deviation Detection in Large Databases. *InKDD*, 1141(50):972–981, 1996 Aug 2.

[3] J. Han and M. Kamber. Data mining: concepts and techniques. *2nd. University of Illinois at Urbana Champaign: Morgan Kaufmann*, 1141(50): 972–981, 2006.

[4] M. Kantardzic. *DataMining Concepts*. 2003.

[5] B. Saneja and R. Rani. An efficient approach for outlier detection in big sensor data of health care. *International journal of communication systems*, 30(17), 2017 Nov 25. doi:10.1002/dac.3352.

[6] V. Chandola, A. Banerjee, and V. Kamber.

Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2019 Ju 30. doi:10.1002/dac.3352.

[7] DS. Shukla, AC. Pandey, and A. Kulhari. Outlier detection: A survey on techniques of WSNs involving event and error based outliers. *In2014 Innovative Applications of Computational Intelligence on Power, Energy, and Controls with their impact on Humanity (CIPECH)*, 41(3):113–116, 2014 Nov. doi:10.1109/CIPECH.2014.7019101.

[8] G. Lin, L. Xin, H. Feng, and L. Ying. A new outlier detection algorithm and its application in intelligent transportation system. *In2014 IEEE 7th Joint International Information Technology and Artificial Intelligence Conference*, pages 442–445, 2014 Dec 20. doi:10.1109/ITAIC.2014.7065088.

[9] X. Zhou, P. Zhao, Y. Liu, and Z. Cui. Semi-supervised Based Training Set Construction for Outlier Detection. In *In2013 International Conference on Cloud Computing and Big Data*. IEEE, 2013 Dec. ISBN 978-1-5386-7178-8. doi:110.1109/CLOUDCOM-ASIA.2013.96.

[10] JP. Vaumi, BO. Yenke, N. Bame, and I. Sarr. Outliers Detection in One Dimensional Meteorological Data Stream. In *In2018 14th International Conference on Signal-Image Technology and Internet-Based Systems (SITIS)*, pages 574–579. IEEE, 2018 Nov 26. doi:10.1109/SITIS.2018.00093.

[11] B. Krawczyk, LL. Minku, J. Gama, and J. Stefanowski M. Woźniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132–56, 2017 Sep. doi:10.1016/j.inffus.2017.02.004.

[12] MM. Breunig, HP. Kriegel, RT. Ng, and J. Sander. LOF: identifying density-based local outliers. In *InProceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000 May 16.

[13] D. Pokrajac, A. Lazarevic, and LJ. Latecki. LOF: identifying density-based local outliers. In , pages 504–515. In2007 IEEE symposium on computational intelligence and data mining, 2007 Mar 1. doi:10.1109/CIDM.2007.368917.

[14] M. Salehi, C. Leckie, JC. Bezdek, T. Vaithianathan, and X. Zhang. Fast memory efficient local outlier detection in data streams. *IEEE Transactions on Knowledge and Data Engineering*, 28(12):3246–60, 2016 Aug. doi:10.1109/TKDE.2016.2597833.

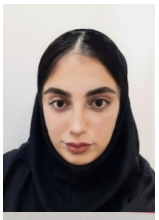
[15] GS. Na, D. Kim, and H. Yu. Dilof. Effective and memory efficient local outlier detection in data streams. *In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1993–2002, 2018 Jul 19. doi:10.1109/TKDE.2016.2597833.

[16] JW. Huang, MX. Zhong, and BP. Jaysawal. Tadilof: time aware density-based incremental local outlier detection in data streams. *Sensors*, 20(20):5829, 2020 Oct 15. doi:doi:10.3390/s20205829.

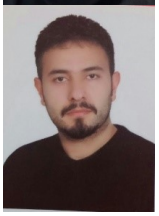
[17] R Alsini, O. Alghushairy, X. Ma, and T. Soule. A grid partition-based local outlier factor for data stream processing. *In Advances in Artificial Intelligence and Applied Cognitive Computing*, 20(20):1047–1060, 2021. doi:10.1007/978-3-030-70296-0_83.

[18] Y. Yang, L. Chen, and C. Fan. ELOF: fast and memory-efficient anomaly detection algorithm in data streams. *Soft Computing.*, 25(6):4283–94, 2021 Mar. doi:10.1007/s00500-020-05442-1.

[19] Datasets. `https://archive.ics.uci.edu/ml/index.php`, Date Accessed: June 29, 2019.



Hadid Mollashahi completed her M.Sc in IT engineering, majoring in computer networks, from the University of Birjand. She received B.Sc degree in IT engineering in 2015. Her research interests and thesis topic include stream data mining, outlier detection, big data analytics, and clustering.



Dr. Hamid Saadatfar is currently an assistant professor of the Computer Engineering Department at the University of Birjand. He received his B.Sc., M.Sc., and Ph.D. degrees from Ferdowsi university of Mashhad in 2007, 2009, and 2014, respectively. His research interests include Parallel and Distributed Processing (Cluster, Grid, and Cloud Computing), Data Mining and Machine Learning, Big Data Analysis (Data Mining Methods for Big Data), and Power-aware Computing.

Dr. Hamed Vahdat-Nejad is currently an associate professor at the computer engineering department of the University of Birjand. He was a visiting professor at the Superior University in Lahore in the Summer of 2018 and at the Daffodil International University in Dhaka in the Summer of 2017. He received his Ph.D. from the computer engineering department of the University of Isfahan in 2012, his master's degree from the Ferdowsi University of Mashhad in 2007, and his bachelor's degree from the Sharif University of Technology in 2004. He was a research scholar with the Middleware laboratory at the Sapienza University of Rome in the 2011-2012 period. Currently, his research is focused on smart city & IoT, Crowd-sourcing, and text processing. He has (co)published more than 50 papers in conferences and journals and led the Pervasive & Cloud computing Lab at the University of Birjand. He has served as the chairman of the 1st & 2nd International Workshop on Context-aware Middleware for Ubiquitous Computing Environments, "3rd, 4th, and 5th International Workshop on Pervasive and Context-aware middleware" as well as the 1st conference on healthcare computing systems and technologies. He has served as a TPC member for many conferences. Currently, he is an associate editor for Elsevier Computers and electrical engineering Journal.