Outlier and Trend Detection Using Approximate Median and Median Absolute Deviation

Gagandeep Singh, Suman Kundu
Department of Computer Science and Engineering, Indian Institute of Technology Jodhpur,
Jodhpur - 342037, India Email: {singh.23, suman}@iitj.ac.in

Abstract—**In this modern era of technologies of scale, vast amounts of data are generated both by users and machines every day. This data comes as streams that may contain outliers. Detecting those outliers can be helpful in many ways, such as machine failures due to overload. Similarly, trends in social media posts are also outliers, and detecting them at different levels has great benefits. The current paper proposes an algorithm to approximate median and median absolute deviation from a stream of numerical values. The algorithm takes a fixed number of memory spaces and linear to the size of the memory. The median and median absolute deviation are then used to detect outliers and multi-level trends without being prone to noise in the data. Experimental results with CPU usage benchmark data and Twitter post data show the effectiveness of the proposed algorithms.**

I. Introduction

Detecting outliers [1, 2, 3] and trends [4, 5] has been important tasks for data mining and computer science. It became more relevant in the past decades [6, 7, 8, 9, 10] due to the amount of crucial data generated in real-time both by users and machines. For example, an estimate of around 7000 tweets was made every minute in the year 2020, out of which around 700 million tweets were sent about elections around the globe [11]. Twitter runs over 150 thousand applications and launches 130 million containers each day [12] to manage such workload. Similarly, a company like Bloomberg, which enables real-time market analysis to its clients, provides access to 35 million instruments across all asset classes [13]. This data is more than enough to realize the scale at which the modern internet works. This also rules out the possibility of manually detecting the trends and anomalies in an ever-evolving stream of data.

Recent events like capital riots in the US and harmful effects of hate speech [14] makes trend detection and that too at many levels an inevitable task. Furthermore, detecting outliers in machine-generated data early on holds great significance. It is evident from the recent outages suffered by Google. It is worthwhile to note that when the impact was visible to various services due to issues with automated resource allocation, the app status page did not update until 30 minutes [15]. This helps realize the importance of developing fast algorithms capable of detecting outliers in machine-generated data to avoid the negative impact.

Approaches like clustering techniques [16, 17, 18, 19] may work well with static data. However, with ever-evolving data streams, such techniques will require infinite memory, which is practically impossible. In addition to that, approaches which use statistics like mean and standard deviation or their variants to detect outliers are prone to noise [20] and suffer from numerical issues [21]. Also, this requires the underlying data to be normally distributed, which is often not the case [20].

The facts, as mentioned earlier, motivate to development of algorithms that can quickly detect outliers and trends in an ever-evolving stream of data without being prone to noise by utilizing limited resources. In this paper, we present two such algorithms, one to detect outliers in numerical data and another to detect trends at multiple levels in text data streams. The main contribution of the paper can be summarized as follows:

- First, we propose a new median and median absolute deviation estimator from a numerical data stream. These estimations are then used in a new algorithm to detect outliers from numerical data streams. We showed experimentally that the proposed algorithm could detect outliers from the CPU usage data of Numenta Anomaly Benchmark [22].
- Secondly, we propose an algorithm to detect trends in the text data stream. Here we tokenized the text into different sets of words, and their frequencies are treated as the numerical data stream. We used the previously proposed anomaly detection algorithm to identify trending posts. Our algorithm can detect trends in the stream in different levels of abstraction simultaneously.
- Finally, we demonstrate the capabilities of the algorithms experimentally.

The rest of the paper is organized as follows. In Section II, we concretely define the problems of outlier and multi-level trend detection. Sections III and IV we propose our algorithms and analyze their performance and efficacy. Section V discusses the related works, and wherever possible, we present a comparison with ours. Experiments and results are reported in Section VI. Finally, we concluded the findings and limitations in Section VII.

II. Problem Definition

The problems of outlier detection and multi-level trend detection from the data stream are defined here in this section.

A. Outlier detection in numerical data stream

Let us consider that $\mathbf{X}$ denotes an ever-evolving stream of numerical values, and $\mathbf{X}_t$ denotes the t^{th} data point in $\mathbf{X}$. We define the problem of outlier data detection as follows,

Definition 1 (Outlier)**.** *Any data point,* $\mathbf{X}_t$ *is said to an outlier if it follows a distribution significantly different from that of the data seen in the past.*

The problem of outlier detection is to identify all such outliers from the numerical data streams online. It is worthwhile to mention that the same data point can be an outlier at one point in time but may not be an outlier in the future [23]. In other words, $\mathbf{X}_t$ may or may not be an anomaly depending on the value of t even though it attains the same value. Note that, from here onwards, we will be using the terms anomaly and outlier interchangeably with no difference in their meaning.

B. Multi-level trend detection in textual data stream

Consider $\mathbf{D}$ as a stream containing text documents and $\mathbf{D}_{1\ldots t}$ be the set of documents till the t^{th} time stamp. Further, let W_l be the set of l-length combination of words. For example, $\{is, are\}$ belongs to W_2 and $\{is, are, am\}$ belongs to W_3. We say that W_l may contain trends at level l.

Definition 2 (Trend at l^{th} level)**.** *Any word combination in* W_l *will said to be a trend if the frequency of occurrence of that combination in* $\mathbf{D}_{1\ldots t}$ *is a numerical outlier.*

The problem of multi-level trend detection is to detect trends from text data stream for different values of l. It should be noted that the frequency of all word combinations in W_l is taken into account to determine the outlier frequencies and their associated word combinations. In simple terms, those word combinations are marked as trends which have too high frequencies compared to other word combinations in the same level. This interpretation also ensures to mark trends which suffer sudden change in frequencies compared to others.

III. Algorithm for Outlier Detection

In this section, we present algorithms for detecting outlier in a numerical data stream. We first approximately estimate the median and median absolute deviations (MAD) from the stream and then detect the outlier using the same.

A. Mean and MAD Estimation

Algorithm 1 shows the approximate median estimator. It selects an element from the stream with probability $\frac{1}{2}$ and keeps it in an ordered multiset S. S is the sampled items from which the median and MAD are estimated on the query. We made the set S as a multiset to keep the same numerical value multiple times as the stream progresses. A point to note here is that the primary objective of the algorithm is to work with univariate numerical data. Hence, the ordered set S is ordered based on the value of the data point. The algorithm returns the median and MAD estimated from the sampled values. The size of S is restricted with a threshold θ. If $length(S)$ is equal to θ while inserting a new element, then we remove one of the elements with $\frac{1}{length(S)}$ probability to make space for the incoming data. This allows for adding newer values of the stream to S by removing an old one. The time and space complexity of the above algorithm depends upon the value of θ. The time complexity of the median query is constant for ordered multiset S. On the other hand, once the median is calculated, MAD can be calculated by a single pass of the sample. Hence, the complexity is $O(\theta)$.

Algorithm 1: Approximate Median and MAD

```
Input: $\mathbf{X}_t \rightarrow t^{th}$ numerical value in the stream; $S \rightarrow$ an ordered
       multiset of sampled data points from $\mathbf{X}_{1\ldots t-1}$; $\theta \rightarrow$ the
       maximum size of $S$;
Output: On query returns: $\mathbf{M} \rightarrow$ the approximate median of
        $\mathbf{X}_{1\ldots t}$; $\mathbf{MAD} \rightarrow$ the approximate median absolute
        deviation of the stream seen so far;
// Sampling of the stream
p ← random(0, 1);
if p $\geq \frac{1}{2}$ then
    if $length(S) \geq \theta$ then
        $remove(S)$;
        // $remove(.)$ function removes one element
           randomly from (.)
        ;
    end
    $S$.add($X_t$);
end
// Query
$\mathbf{M} \leftarrow get(S, \frac{length(S)}{2})$;
$\mathbf{MAD} \leftarrow 0$;
i ← 0;
while i $\leq length(S) - 1$ do
    $\mathbf{MAD} \mathrel{+}= get(S, \mathbf{i}) - \mathbf{M}$;
    i += 1;
end
$\mathbf{MAD} \leftarrow \frac{\mathbf{MAD}}{length(S)}$;
return $\mathbf{M}, \mathbf{MAD}$
```

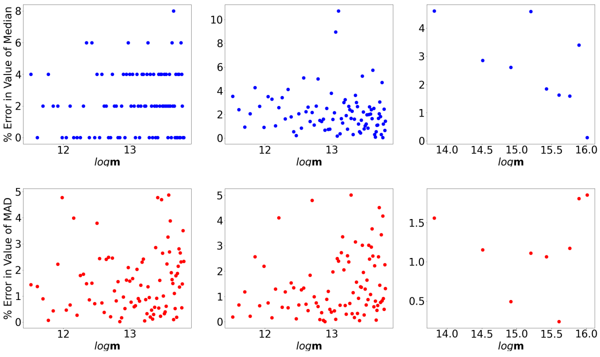

Fig. 1. Percentage error in the values of median (top) and MAD (bottom) computed by Algorithm 1 for synthetically generated numerical data stream.

It is trivial to observe that with the increase in the size of S, the approximations of median and median absolute deviation will get better. However, it is also important to answer the question that, “How large θ should be to get reasonable estimates of the above two statistics of the stream?”. In [24], $\theta \geq \frac{\log 2\delta^{-1}}{2\epsilon_2^2}$ was used to compute an $\epsilon = \epsilon_1 + \epsilon_2$ approximate of ϕ-quantile of a dataset of size N. We may use this result in our algorithm. However, since we are dealing with data streams instead of a static data set, we may not be able to leverage the opportunity to look at a single element more than once, in which case sampling with replacement may not be possible. Therefore, our algorithm simplifies the above task and selects an element based on the tossing of an unbiased coin. In order to analyze the error percentage of the estimated mean and MAD with the fraction of the number of elements in the stream as θ, we ran Algorithm 1 for different values of the number of elements in the stream $(\mathbf{m})$. The result is shown in Figure 1. The images in the top row show the percentage of error in the value of median, whereas the bottom row shows the error in the MAD. For all three experiments, we used $\theta = 1000$. Different plots show results for different range of values for $\mathbf{X}_t$, from the first column till third column these values are in the range of $[0, 100]$, $[0, 10^6]$, and $[0, 35 \times 10^8]$ respectively. X-axis shows the logarithmic value of the total number of data elements in the stream. The error percentage is calculated based on the actual mean (or MAD) with the approximated mean (or MAD) obtained from the proposed Algorithm 1. It is observed from the result is that the error is not more than 4 % for the majority of the experiments.

Algorithm 2: Detect Anomalies in Numerical Stream

```
Input: $\mathbf{X}_t \rightarrow t^{th}$ numerical value in the stream; $S \rightarrow$ an ordered
       multiset of sampled data points; $c$ & $k \rightarrow$ the threshold for
       detecting anomalies and noise removal; $\theta \rightarrow$ the maximum
       size of $S$;
Output: $True \rightarrow$ when $\mathbf{X}_t$ is anomaly; $False \rightarrow$ when $\mathbf{X}_t$ is
        neither anomaly nor noise; $None \rightarrow$ when $\mathbf{X}_t$ is detected
        as noise.
$\mathbf{M}, \mathbf{MAD} \leftarrow Algorithm1.Query()$;
if $\mathbf{X}_t \geq \mathbf{M} + k \times c \times \mathbf{MAD}$ then
    return $\mathbf{None}$;
end
if $X_t \geq \mathbf{M} + c \times \mathbf{MADor}X_t \leq \mathbf{M} - c \times \mathbf{MAD}$ then
    return True;
else
    return False;
end
```

B. Outlier Detection

The proposed algorithm, shown in Algorithm 2, detects whether an incoming element $\mathbf{X}_t$ is an anomaly or not. In its core, Algorithm 2 uses Algorithm 1 to estimate the mean and MAD. Variables c and k are user-defined parameters that control the tolerance of the anomaly and noise, respectively. In other words, if an incoming element is away from the median by c factor of MAD but less than $k \times c$ factor of MAD, we call the data point an anomaly. It is worthy to note here that since the median has a breaking point of 0.5 [20], or in simple words, it is not corrupted by noise in the data easily as opposed to the mean, which has a breaking point of 0 chances of noise close to the mean is low. Hence, we consider a data point noise if it is $k \times c \times MAD$ away from the median. Algorithm 2 can be executed in constant time if the mean and MDA are known. Thus, the time and space complexity of Algorithm 2 is the same as Algorithm 1 with some constant overhead incurred due to anomaly and noise checks.

Algorithm 3: Detect Trends in a stream of Text

```
Input: $\mathbf{D}_t \rightarrow t^{th}$ numerical value in the stream; $S \rightarrow l_m \times N_h$
       size hash table mapping the level to its associated ordered
       multiset of frequencies of word combinations $\mathbf{W}_l$ from
       $\mathbf{D}_{1...t-1}$; $\mathbf{freqs} \rightarrow$ a hash table mapping levels to the
       maximum frequency of a word combination to that level;
       $c \rightarrow$ the threshold for detecting anomalies; $\mathbf{h} \rightarrow$ a hash
       function which maps a string to an integer; $\mathbf{l}_m \rightarrow$, the
       maximum level to be considered for trend detection; $\theta \rightarrow$ the
       maximum size of the samples;
Output: $\mathbf{results} \rightarrow$ set of found trends;
1  $W \leftarrow tokenize(\mathbf{D}_t, [1, 2, ..., \mathbf{l}_m])$;
   // $tokenize(.,.)$ function tokenized the first
      input into word combinations of lengths
      passed by the 2nd argument
2  ;
3  for $\mathbf{W}_l \in W$ do
4      for $wordcomb$ in $\mathbf{W}_l$ do
5          $h \leftarrow \mathbf{h}(wordcomb)$ Update $\mathbf{freqs}[l][h]$ with the
             maximum frequency.;
6          $S[l] \leftarrow Algorithm1.Sample(\mathbf{freqs}[l][h], \theta)$
7      end
8  end
9  for $l \in$ freqs do
10     for $hash$ in $\mathbf{freqs}[l]$ do
11         if $Algorithm2(S[l], c, k = 2, \theta) = True$ then
12             $\mathbf{results}[l].add(hash)$;
13         end
14     end
15 end
```

IV. Algorithm for Trend Detection

Algorithm 3 shows the proposed algorithm for trend detection from stream of text documents such as Tweets. A document D_t (t_{th} document in the stream) is first tokenized into combinations of words. We refer the size of the token as level and the max level (l_m) to consider is an user input parameter to the algorithm. These word combinations are then hashed for ease of use and the maximum frequency is captured by the algorithm for detecting the trends. Furthermore, the trends are detected by utilising the numerical outlier detection algorithm discussed in Section III from the frequency stream. It should be noted that inverse mapping of $\mathbf{h}$ in Algorithm 3 is maintained in order to retrieve the original trends, although this inverse mapping will not be utilised during the execution of the algorithm itself.

a) Complexity:: The lines 3 to 8 will run for all possible combinations of words of length from 1 till l_m. If we consider the average number of words in document D_t is N_w then the complexity of these loops will be $O((N_w)^{l_m})$. On the other hand the running time complexity of the lines 9 to 15 is equal to l_m times the size of the range of the hash function, N_h say. Thus the time complexity of Algorithm 3 is $O((N_w)^{l_m} + l_m \times N_h)$. The space complexity will be $O(l_m \times (\max(\theta, N_h))$.

V. Discussion and Related Work

This section discusses the studies conducted so far in the area of outlier and trend detection and wherever possible we have made a comparison with ours.

A. *Outlier Detection*

The most basic though simple approach is ‘three-sigma rule’ for detecting anomalies in a data set. It uses the distance from the mean value as the metric to decide whether a given data point is anomalous. Specifically, if a data point is more than 3 standard deviations (SD) away from the mean, it is marked as an outlier. Despite the assumption that the underlying data is normally distributed is impractical [20], this technique is efficient. Specifically, it requires only one pass over the data set and consumes O(1) extra memory. In a data stream, running averages and SD can be computed to implement the above technique. However, the mean has a breaking point of 0 compared to the median, which has a breaking point of 0.5. That is the mean can easily be corrupted with noise, but the median can handle at most 50% of noisy data. Other approaches along similar lines are Exponentially Weighted Moving Averages (EWMA) [25], which exponentially reduces the weight of data points as time passes, i.e., it gives more weight to the recent values than the older ones. Probabilistic EWMA was proposed by [26] to solve the limitation of EWMA’s response to abrupt changes in the data stream. The issues with mean still persist when we use a simple moving average, EWMA or PEWMA. Our algorithms utilize a reasonable approximation of the median coupled with the median absolute deviation, which solves the problem of corruption due to noise with moving averages & SD.

Grubb’s Test [27, 28] was proposed to detect a single anomaly in a given data set. It defines a null hypothesis test by assuming the underlying data to be normally distributed. It also uses the mean and variance of the data, making it prone to the limitations of these statistics, i.e., easily getting corrupted due to noise. In addition to that, it is capable of only detecting the largest anomaly in a given data set. Tests like Extreme Studentised Deviate (ESD) [29], Seasonal-ESD [20], Seasonal-Hybrid-ESD [20] build on top of Grubb’s test by running the hypothesis test a given a number of times. Hence, they are capable of detecting only a given amount of anomalies in a stream and that too by requiring more than one pass of data. Our algorithms are capable of detecting multiple anomalies in a given data stream in a single pass, and they do not assume anything about the underlying data distribution.

Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) [21] detects anomalies in a data set/data stream using a height-balanced B-Tree. Each node of the tree stores a cluster feature which is a triplet of the number of data points in that cluster and their linear and squared sums. It periodically scans the tree and flags the low-density leaves as outliers. LOF [30] is one of the most popular algorithms for anomaly detection. It computes scores based on k-distance, reachability distance, and local reachability density to mark a data point as anomaly. This approach is well suited for static data sets where all data points are already available. However, such approaches may not work in a data stream due to the high computational resource requirement. iLOF [31] is an improvement over LOF but still suffers from high memory requirements for retaining the previous data points.

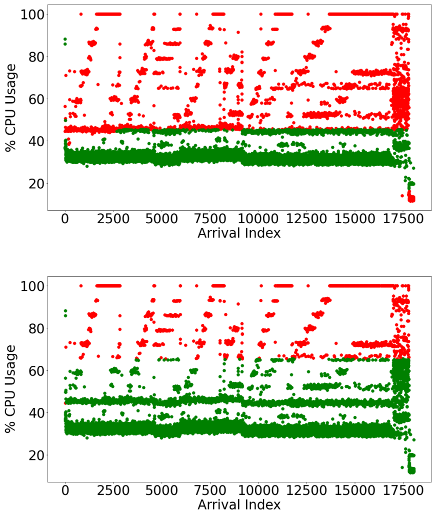

Fig. 2. Results from Algorithm 2 on CPU usage data [22]. The red dots denote the anomalies flagged by the algorithm.

B. *Trend Detection*

First Story Detection [32] is a problem closely related to trend detection, where algorithms attempt to find the origin of an event. Our algorithms detect trends when they achieve a frequency in a particular range. Anything above the upper bound of that is considered as noise. Clustering techniques have been used [16] to group the incoming documents of a stream in groups with respect to their similarity by representing them in a vector space. The incoming documents, which differ significantly from the previous documents, are reported as the origins of the event highlighted by that document. This approach may not be efficient when applied to a never-ending stream due to high memory requirements for storing the previous records. Moreover, verifying if the new document is an anomaly or not, any existing techniques will require O(N) time where N is the total number of documents or O(K) processors, where K is the number of groups. Our algorithm does not require storing the entire set of documents. However, it does store the maximum frequency of word combinations generated from the incoming document at each level.

Approaches like [33, 34] attempt to solve the memory problem by either using limited data like hashtags from tweets or titles of blog posts which reduces the quality of the results obtained. In [32], the memory problem has been solved by developing a hashing technique to store the frequencies of the hash of only the most popular words. However, the measures like z-scores used in [32] assume that the data is normally distributed, which is rarely the case [20]. We use approximate median and median absolute deviation to detect outlier frequencies and that too at multiple levels, which is not the case in any previous works as per our best knowledge. This makes our algorithm less prone to noise in the data stream.

VI. Experiments and Results

In this section, we present and analyze the results of our experiments. Algorithms are implemented in Python 3.x on Google colab, a freely available infrastructure for research. We show the capabilities of our algorithms through experimental results and show how they can be used as an additional tool in a system independently or along with other anomaly detection techniques. We perform two different experiments to understand the capabilities of the proposed algorithm, first with the anomaly benchmark data of CPU usage [22] for identifying numerical anomalies and second with the Twitter data [35] to extract trends out of Tweets.

We used CPU usage data from Numenta Anomaly Benchmark [22] for evaluating Algorithm 2. Different experiments are performed for different values of c and θ. Figure 2 shows the results. The x-axis shows the arrival index, and the % of CPU usage is shown in the y-axis. The top plot shows the results for $c = 1.3$ and $\theta = 200$ while bottom plot shows results for $c = 2$ and $\theta = 1000$. It is observed from the plots that abnormally high CPU usages are marked as outliers for both configurations. Note that, in the top plot, the 40% to 50% band of CPU usage is initially marked as an anomaly, but later the algorithm is self-corrected and considered as normal CPU usage. Further to that, abnormally low CPU usages in the latter part are also marked as anomalies that may reflect a system failure resulting in low CPU usages compared to the regular pattern. With the increase in the value of c the same is not visible in the bottom side plot.

Algorithm 3 is evaluated on tweets from May 2019, which were available on the internet archive [35]. In the experiment, we used around 12000 tweets, and the value of θ and c was kept as 100 and 1.5 respectively. k was taken as 3 in this case. We observed that our algorithm was able to capture trends with very small frequencies, for example, $billboard$, having a very low frequency of 6 was captured at level 1, and $1VoteBBMAsTopSocial\ EXOVotingTeam$ $favorite1$ with a frequency of 3 was also captured at level 3. The result indicates the happening of Billboard Music Awards. One may note that the space between words was removed as a pre-processing while creating the hash values. Further to that, non-trending but high-frequency word combinations were automatically removed. For example, RT was one of the most used words but was not captured. This indicates that our algorithms are not easily corrupted by spam and noise in the data. The word cloud generated from the results of the proposed algorithm is shown in Figure 3 for reference.

VII. Conclusion

This paper proposed efficient algorithms for outlier and trend detection from streams by utilizing a new approximate median and median absolute deviation. The proposed algorithms do not easily break by noise in the data stream, unlike the mean and standard deviation based outlier and trend detection algorithms. We showed experimentally that the proposed algorithm works as expected and can detect anomalies and trends from real-world data streams.

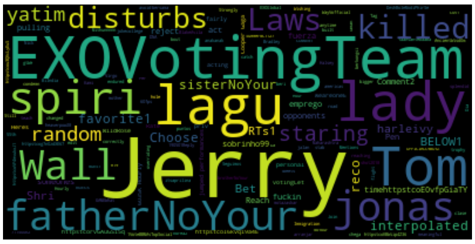

Fig. 3. Word cloud of trends captured by proposed trend detection Algorithm 3 from tweets of May, 2019.

One limitation of Algorithm 1 is that it depends on specifying θ. This impacts the algorithm's performance significantly as the approximation of median and median absolute deviations will depend upon it. However, experimentally it is found to give good results when we kept $\theta = 1000$. The error was within a range of 4%. The parameters c and k of Algorithm 1 are user-defined parameters that specify the tolerance level for anomalies and noisy data. Our experiment shows promising results for the value of c between 1 to 3 and $k = 2$. However, it may depend on the problem at hand, and the user can select based on some trials.

The execution time of Algorithm 3 depends on the choice of the maximum level and the average length of the documents received in the stream. To obtain a reasonable execution time, one should keep the value of the maximum level low when the number of words in the document is significant.

Through the experiments on benchmark data, we showed the applicability of the proposed algorithm. As a future direction, a comparison with existing outlier detection algorithms may be explored. The presented algorithms may also work simultaneously to complement the existing outlier detection algorithm by detecting the early sign of anomalies in the stream.

References

[1] M. Salehi, C. Leckie, J. C. Bezdek, T. Vaithianathan, and X. Zhang, "Fast Memory Efficient Local Outlier Detection in Data Streams," *IEEE TKDE*, vol. 28, no. 12, pp. 3246–3260, 2016.

[2] L. Cao, D. Yang, Q. Wang, Y. Yu, J. Wang, and E. A. Rundensteiner, "Scalable distance-based outlier detection over high-volume data streams," in *Proc. of International Conf. on Data Engineering*, 2014, pp. 76–87.

[3] X. Qin, L. Cao, E. A. Rundensteiner, and S. Madden, "Scalable kernel density estimation-based local outlier detection over large data streams," in *Advances in Database Technology*, vol. 2019-Mar, 2019, pp. 421–432.

[4] S. Malakar, S. Goswami, and A. Chakrabarti, "An online trend detection strategy for twitter using mann–Kendall non-parametric test," in *Lecture Notes in Networks and Systems*. Springer, 2018, vol. 11, pp. 185–193.

[5] H. Abe and S. Tsumoto, "Trend detection from large text data," in *Proc. of IEEE International Conference on Systems, Man and Cybernetics*, 2010, pp. 310–315.

[6] S. Cai, S. Hao, R. Sun, and G. Wu, "Mining recent maximal frequent itemsets over data streams with sliding window," *International Arab Journal of Information Technology*, vol. 16, no. 6, pp. 961–969, nov 2019.

[7] S. Cai, S. Li, G. Yuan, S. Hao, and R. Sun, "MiFI-Outlier: Minimal infrequent itemset-based outlier detection approach on uncertain data stream," *Knowledge-Based Systems*, vol. 191, p. 105268, mar 2020.

[8] M. Parto, C. Saldana, and T. Kurfess, "Real-time outlier detection and Bayesian classification using incremental computations for efficient and scalable stream analytics for IoT for manufacturing," in *Procedia Manufacturing*, vol. 48. Elsevier B.V., jan 2020, pp. 968–979.

[9] S. Yoon, J. G. Lee, and B. S. Lee, "NETS: Extremely fast outlier detection from a data stream via set-based processing," in *Proceedings of the VLDB Endowment*, vol. 12, no. 11, 2018, pp. 1303–1315.

[10] G. Shevlyakov and M. Kan, "Stream Data Preprocessing: Outlier Detection Based on the Chebyshev Inequality with Applications," in *Conference of Open Innovation Association, FRUCT*, vol. 2020-April. IEEE Computer Society, apr 2020, pp. 402–407.

[11] T. McGraw, "Spending 2020 together on twitter," https://blog.twitter.com/en_us/topics/insights/2020/spending-2020-together-on-twitter.html, accessed: 2021-01-31.

[12] M. Hashemi, "The infrastructure behind twitter: Scale," https://blog.twitter.com/engineering/en_us/topics/infrastructure/2017/the-infrastructure-behind-twitter-scale.html, accessed: 2021-01-31.

[13] "Market data — bloomberg professional services," https://www.bloomberg.com/professional/product/market-data/, accessed: 2021-01-31.

[14] J. Uyheng and K. M. Carley, "Bots and online hate during the covid-19 pandemic: case studies in the united states and the philippines," *Journal of Computational Social Science*, vol. 3, no. 2, pp. 445–468, 2020.

[15] K. Kareem, "Google cloud outage: a lesson in reducing mean time to detect," https://blog.catchpoint.com/2019/06/03/google-cloud-outage/, accessed: 2021-01-31.

[16] J. Allan, V. Lavrenko, D. Malin, and R. Swan, "Detections, bounds, and timelines: Umass and tdt-3," in *Proc. of Topic Detection and Tracking Workshop*, 2000.

[17] E. F. Cabral and R. L. Cordeiro, "Fast and Scalable Outlier Detection with Sorted Hypercubes," in *Proc. of International Conference on Information and Knowledge Management*. New York: ACM, 2020, pp. 95–104.

[18] S. Li, M. Shao, and Y. Fu, "Multi-view low-rank analysis with applications to outlier detection," *ACM Transactions on Knowledge Discovery from Data*, vol. 12, no. 3, 2018.

[19] Y. H. Kuo, Z. Li, and D. Kifer, "Detecting outliers in data with correlated measures," in *Proc. of International Conference on Information and Knowledge Management*. ACM, 2018, pp. 287–296.

[20] J. Hochenbaum, O. S. Vallis, and A. Kejariwal, "Automatic anomaly detection in the cloud via statistical learning," *arXiv preprint arXiv:1704.07706*, 2017.

[21] Wikipedia contributors, "Birch - Wikipedia, the free encyclopedia," 2021, [accessed 2021-01-27]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=BIRCH&oldid=1000989569

[22] S. Ahmad, A. Lavin, and S. P. et. al, "numenta/nab: v1.1," Dec. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3571294

[23] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, 2009.

[24] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "Approximate medians and other quantiles in one pass and with limited memory," *ACM SIGMOD Record*, vol. 27, no. 2, pp. 426–435, 1998.

[25] J. M. Lucas and M. S. Saccucci, "Exponentially weighted moving average control schemes: properties and enhancements," *Technometrics*, vol. 32, no. 1, pp. 1–12, 1990.

[26] K. M. Carter and W. W. Streilein, "Probabilistic reasoning for streaming anomaly detection," in *2012 IEEE Statistical Signal Processing Workshop (SSP)*, 2012, pp. 377–380.

[27] F. E. Grubbs *et al.*, "Sample criteria for testing outlying observations," *Annals of mathematical statistics*, vol. 21, no. 1, pp. 27–58, 1950.

[28] F. E. Grubbs, "Procedures for detecting outlying observations in samples," *Technometrics*, vol. 11, no. 1, pp. 1–21, 1969.

[29] B. Rosner, "On the detection of many outliers," *Technometrics*, vol. 17, no. 2, pp. 221–227, 1975.

[30] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.

[31] D. Pokrajac, A. Lazarevic, and L. J. Latecki, "Incremental local outlier detection for data streams," in *2007 IEEE Symposium on Computational Intelligence and Data Mining*, 2007, pp. 504–515.

[32] E. Schubert, M. Weiler, and H.-P. Kriegel, "Signitrend: scalable detection of emerging topics in textual streams by hashed significance thresholds," in *Proc. of 20th ACM SIGKDD*, 2014, pp. 871–880.

[33] F. Alvanaki, S. Michel, K. Ramamritham, and G. Weikum, "See what's enblogue: real-time emergent topic identification in social media," in *Proc. of 15th International Conference on Extending Database Technology*, 2012, pp. 336–347.

[34] M. Platakis, D. Kotsakos, and D. Gunopulos, "Searching for events in the blogosphere," in *Proceedings of the 18th international conference on World wide web*, 2009, pp. 1225–1226.

[35] @sketch_the_cow, "Twitter stream archive," https://archive.org/details/archiveteam-twitter-stream-2019-05, accessed: 2021-01-31.