Atypicity Detection in Data Streams: a Self-Adjusting Approach

Alice Marascu, Florent Masseglia

ATYPICITY DETECTION IN DATA STREAMS: A SELF-ADJUSTING APPROACH

ALICE MARASCU AND FLORENT MASSEGLIA

INRIA Sophia-Antipolis
AxIS Project-Team
2004 route des lucioles - BP 93
06902 Sophia-Antipolis

ABSTRACT. Outlyingness is a subjective concept relying on the isolation level of a (set of) record(s). Clustering-based outlier detection is a field that aims to cluster data and to detect outliers depending on their characteristics (*i.e.* small, tight and/or dense clusters might be considered as outliers). Existing methods require a parameter standing for the "level of outlyingness", such as the maximum size or a percentage of small clusters, in order to build the set of outliers. Unfortunately, manually setting this parameter in a streaming environment should not be possible, given the fast time response usually needed. In this paper we propose WOD, a method that separates outliers from clusters thanks to a natural and effective principle. The main advantages of WOD are its ability to automatically adjust to any clustering result and to be parameterless.

1. INTRODUCTION

Atypical behaviours are the basis of a valuable knowledge in domains related to security (*e.g.* fraud detection for credit card [3], cyber security [9] or safety of critical systems [12]). Atypicity generally depends on the isolation level of a (set of) records, compared to the dataset. One possible method for finding atypic records aims to perform two steps. The first step is a clustering (grouping the records by similarity) and the second step is the identification of clusters that do not correspond to a satisfying number of records. Actually, atypical events (or outliers) might be indicative of suspicious data such as skewed or erroneous values, entry mistakes or malicious behaviours. A malicious behaviour can be detected as an outlier in datasets such as transactions in a credit card database or records of usage on a web site.

To the best of our knowledge, outlier detection always relies on a parameter, given by the end-user and standing for a "degree of outlyingness" above which records are considered as atypical. For instance, in [17], a *distance-based* outlier is an object such that a *user-defined fraction* of dataset objects have a distance of more than a *user-defined minimum distance* from that object. In [11], the authors propose a nonparametric clustering process and the detection of outliers requires a *user defined value* k corresponding to the top-k desired outliers.

In this paper we propose WOD (Wavelet-based Outlier Detection), a parameterless method intending to automatically extract outliers from a dataset. In contrast to previous work, our goal is to find the best division of a distribution and to automatically separate values into two sets corresponding to clusters on the one hand and outliers on the other hand. The tail of the distribution is found thanks to a wavelet technique and does not depend on

any user threshold. Our method fits any distribution depending on any characteristic such as distances between objects [17], objects' density [5, 24] or clusters' size [14, 27].

Our framework involves clustering-based outlier detection in data streams. Clustering-based detection of outliers aims to find objects that do not follow the same model as the rest of the data depending on the clusters' size or tightness [14, 27, 11]. This framework will allow us to illustrate our proposal with one of the possible characteristics observed for building a distribution of objects (*i.e.* clusters' size). The choice of data streams is motivated by the specific constraints of this domain. In a data stream environment, data are generated at a very high rate and it is not possible to perform blocking operations. In this context, requesting a parameter such as k, for top-k outliers, or x, a percentage of small clusters, should be prohibited. First, because the user doesn't have enough time to try different values of these parameters for each period of analysis on the stream. Secondly, because a permanent value may be adapted to one period of the stream but it is highly likely to be wrong on the next periods (the data distribution will change, as well as the number or percentage of outliers). For these reasons, detecting outliers should not depend on any parameter and should be adaptive in order to keep the best accuracy all along the stream.

This paper is organized as follows. Section 2 gives an overview of existing works in outlier detection and Section 3 gives a formal definition of our problem.

The first step of our method aims to group the streaming data into clusters. We describe two clustering case studies in Section 4. Section 5 gives the details of WOD and its principle for separating outliers from clusters. Section 6 shows the advantages of WOD through a set of experiments on real Web usage data and Section 7 gives our conclusion.

2. RELATED WORKS

Outlier detection has been extensively studied these past years, since it has a wide range of applications, such as fraud detection for credit card [3], cyber security [9] or safety of critical systems [12]. Those fields of application rely on methods to find patterns which deviate significantly from a well-defined notion of normality. The concept of outlyingness has been studied by statistics [23, 19] where statistical approaches construct probability distribution models under which outliers are objects of low probability [4, 20]. Within the context of intrusion detection, data dimensionality is high. Therefore, to improve overall performances and accuracy, it has become necessary to develop data mining algorithms using the whole data distribution as well as most of data features [17, 1].

In this paper, we focus on clustering-based outlier detection algorithms [17, 26, 6, 11, 15, 10, 13, 24]. Such techniques rely on the assumption that normal points belong to large clusters while outliers either do not belong to any cluster [17, 26] or form very small and tight clusters [14, 27, 11]. In other words, outlier detection consists in identifying among data those that are far from being significant clusters. Depending on the approach, the number of parameters required to run the algorithm can be high and will lead to different outliers. To avoid this, some works return a ranked list of potential outliers and limit the number of parameters to be specified [26, 15, 11]. Let us note that [11] proposes to reduce or to avoid given parameter to the clustering algorithm, while maintaining a parameter regarding the outliers: n the number of required outliers. In this paper, we aim to detect outliers on the basis of clusters characteristics only. Among these characteristics, we have selected the clusters' size. A distribution of the clusters' size combined with our wavelet approach allows cutting the clusters set into two sub-sets, basically corresponding to "big" and "small" clusters. This method would also cut down this set with regard to other characteristics, such as clusters tightness or their number of neighbors (density) for

instance. Applications of wavelet theory in data mining have been studied by [21] and the authors propose a survey on this topic.

In [31], the authors propose a technique for both clustering and outlier detection in static data, based on their previous work [28]. [31] considers a dataset of d-dimensional points. Their goal is to apply a wavelet transform on the feature space. For this purpose, they first partition the original feature space into cells in order to obtain a quantized space (where a point o_k belongs to a cell c_i if o_{k_i} is comprised in the intervals of c_{i_j} for each dimension j). Then, a density function $\rho(c_i)$, based on the number of points contained in each cell, is specified. This density function will allow deciding whether a point is an outlier or not depending on a user threshold.

3. Problem Statement

Clustering is the problem of finding a partition of a data set so that similar objects are in the same part of the partition and different objects are in different parts. A data stream $S = \{S_1, ..., S_i, ..., S_n\}$ is a series of batches S_i, read in increasing order of the indices i. Each batch contains a set of objects $O = \{o_1, ..., o_m\}$.

In this paper, we propose to process the stream of a Web site's usage, batch after batch. Let W be the site being analyzed. We propose to study two clustering problems associated to this data.

3.1. Clustering PHP requests.

In this case, our goal is to extract clusters and detect atypical events among the requests performed to PHP scripts on W. Therefore, our objects will be the parameters given to the PHP scripts on W. We do not propose an intrusion detection framework, since atypical usage does not automatically correspond to malicious behaviours. Nevertheless, we expect our results to be useful for an automatic detection of atypicity in data streams. Section 4.2 describes our principle for clustering such data.

3.2. Clustering navigations.

First, we need to define a navigation sequence as the series of URLs requested by a user. This definition will use the definition of itemsets from [2].

Definition 1. *Let $\mathcal{I} = i_1, i_2, ..., i_n$ be a set of items. Let $X = i_1, i_2, ..., i_k / k \leq n$ and $\forall j \in [1..k]\ i_j \in \mathcal{I}$. X is called an* **itemset** *(or a $k-$***itemset***). Let $\mathcal{T} = t_1, t_2, ..., t_m$ be a set of times, over which a linear order $<_T$ is defined, where $t_i <_T t_j$ means t_i occurs before t_j. A* **transaction** *T is a pair $T = (tid, X)$ where tid is the transaction's identifier and X is the associated itemset. Associated to each item i in X we have a time-stamp t_i which represents the valid time of occurrence of i in T.*

Definition 2. *A navigation sequence is an ordered list of itemsets denoted by $< n_1, n_2, \ldots, n_n >$, where n_j is an itemset and each item of n_j stands for a URL.*

In this case, a batch is made of k navigation sequences. Each navigation sequence n in the data stream is associated to a client c and n corresponds to the series of requests performed by c on the Web site. In section 4.1 we propose a method for clustering the navigation sequences of a data stream.

3.3. Atypicity detection.

For each case (PHP requests and navigations), our goal is to separate clusters in order to give the list of atypical events. Our principle (based on a multi-resolution analysis) for this parameterless detection is presented in Section 5.

4. CLUSTERING STREAMING USAGE DATA

Our method will process the data stream as batches of equal size. Let $B_1, B_2, ...B_n$ be the batches, where B_n is the most recent batch of transactions. The principle of WOD will be to cluster the content of each batch b in $[B_1..B_n]$ and to detect outliers according to the clusters' size. In this section, we describe an agglomerative clustering principle that has been adapted on two different kinds of data. As we will observe in the experiments, WOD is a self adjusting outlier detection method. Therefore, in order to assess that feature, we want to work on different datasets with different characteristics. The Web usage dataset has rather constant characteristics (size of clusters, content, distribution, etc.). On the other hand, the PHP dataset shows an important variation of usage in terms of distribution of the clusters.

4.1. **Clustering Navigations.**

The general principle of our method can be described as follows: for each batch of transactions, our algorithm extracts the clusters of users (grouped by behavior) and then analyzes their navigations by means of a sequence alignment process. This allows us to obtain clusters of behaviors that represent the current usage of the Web site. In [22] the authors have proposed a method for mining sequential patterns in data streams which is based on sequence alignment. The clustering function of this paper catches and extends this principle. For each cluster c, the aligned sequence corresponding to the content of c gives a summary of c. After processing each batch, we are provided with patterns (the summaries or alignments obtained for the clusters) and their supports (the size of the clusters).

For each batch, the clustering algorithm is initialized with only one cluster which contains the first navigation (the first sequence of the batch). To each cluster s is associated a centroid ς_c (the aligned sequence of the cluster) that summarizes the cluster. WOD will process the batch of sequences in only one scan. During this scan, the following operations are performed:

(1) For each navigation n in the batch, n is compared to each existing centroid. Let c be the cluster such that its centroid ς_c is the most similar to n, then n is inserted into c. If no such cluster has been found, then a new cluster is created and n is inserted in this new cluster. The comparison of n (the navigation sequence) with a cluster c is explained in Subsection 4.1.2.
(2) For each cluster c, the centroid ς_c of c is computed incrementally. This step (detailed in Subsection 4.1.1) is very important, since each sequence s has to be compared to the centroid of each cluster.
(3) At the end of the scan, the centroid of each cluster c will stand for the extracted knowledge since it can be considered as a summary of c.

4.1.1. *Centroid of a Cluster.* The centroid ς_c of cluster c is computed thanks to an alignment technique applied to c. When the first sequence is inserted into c, ς_c is equal to this unique sequence.

The alignment of sequences is based on the definition of [18] and leads to a weighted sequence represented as follows: $SA =< I_1 : n_1, I_2 : n_2, ..., I_r : n_r >: m$. In this representation, m stands for the total number of sequences involved in the alignment. I_p ($1 \leq p \leq r$) is an itemset represented as ($x_{i_1} : m_{i_1}, ...x_{i_t} : m_{i_t}$),where m_{i_t} is the number of sequences containing the item x_i at the p^{th} position in the aligned sequences. Finally, n_p is the number of occurrences of itemset I_p in the alignment. Example 1 describes the alignment process of 4 sequences. Starting from two sequences, the alignment begins with

Step 1 :			
S_1:	<(a,c)	(e)	()>
S_2:	<(a,d)	(e)	(h)>
SA_{12}:	(a:2, c:1, d:1):2	(e:2):2	(h:1):1
Step 2 :			
SA_{12}:	(a:2, c:1, d:1):2	(e:2):2	(h:1):1
S_3:	<(a,b)	(e)	(i,j)>
SA_{13}:	(a:3, b:1, c:1, d:1):3	(e:3):3	(h:1, i:1, j:1):2
Step 3 :			
SA_{13}:	(a:3, b:1, c:1, d:1):3	(e:3):3	(h:1, i:1, j:1):2
S_4:	<(b)	(e)	(h,i)>
SA_{14}:	(a:3, b:2, c:1, d:1):4	(e:4):4	(h:2, i:2, j:1):3

FIGURE 1. Different steps of the alignment method with sequences from example 1

the insertion of empty items (at the beginning, at the end or inside the sequence) until both sequences contain the same number of itemsets.

Example 1. *Let us consider the following sequences: $S_1 =<$ (a,c) (e) $>$, $S_2 =<$ (a,d) (e) (h) $>$, $S_3 =<$ (a,b) (e) (i,j) $>$, $S_4 =<$ (b) (e) (h,i) $>$. The steps leading to the alignment of these sequences are detailed in Figure 1. First, an empty itemset is inserted at the end of S_1. Then S_1 and S_2 are aligned in order to provide SA_{12}. The alignment process is then applied to SA_{12} and S_3. The alignment method goes on processing two sequences at each step.*

At the end of this alignment process, the aligned sequence (SA_{14} in figure 1) is a summary of the corresponding cluster. An approximate sequential pattern representing each cluster can be obtained by specifying k: the number of occurrences of an item in order for it to be displayed. For instance, for the sequence SA_{14} from Figure 1 and $k = 2$ the filtered aligned sequence will be: <(a,b)(e)(h,i)> (corresponding to the items having a number of occurrences greater or equal to k).

The aligned sequence is incrementally updated, each time a sequence is added to its cluster. For that purpose, we maintain a matrix which contains the number of items for each sequence and a table representing the distances between sequences. This is illustrated in Figure 2. Our matrix (left) stores for each sequence the number of occurrences of each item in this sequence. For instance, s_1 is a sequence containing twice the item a. The table of distances stores the sum of similarities ($similMatrix$) between sequences. Let s_{1_i} be the number of occurrences of item i in sequence s_1 and let m be the total number of items. $similMatrix$ is computed thanks to the matrix in the following way :

$similMatrix(s_1, s_2) = \sum_{i=1}^{m} min(s_{1_i}, s_{2_i})$.

For instance, with two sequences s_1 and s_2 in the matrix of Figure 2, this sum is: $s_{1_a} + s_{2_b} + s_{2_c} = 1 + 0 + 1 = 2$.

Sometimes, the alignment has to be refreshed and cannot be updated incrementally. Let us consider a sequence s_n. First, s_n is inserted in the matrix and its distance to the other

Seq	a	b	c
s_1	2	0	1
s_2	1	0	1
$\vdots$			

Seq	$\sum_{i=1}^{n} similMatrix(s, s_i)$
s_1	16
s_2	14
s_n	13
s_3	11
$\vdots$	
s_{n-1}	1

FIGURE 2. Distances between sequences

sequences is computed ($\sum_{i=1}^{n} similMatrix(s_n, s_i)$). s_n is then inserted in the distance table, with respect to the decreasing order of distances values. For instance, in Figure 2, s_n is inserted after s_2. Let r be the rank where s_n is inserted (in our current example, $r = 2$) in c. After inserting s_n, there are two possibilities:

(1) $r > 0.5 \times |c|$. In this case, the alignment is updated incrementally and $\varsigma_c = alignment(\varsigma_c, s_n)$.
(2) $r \leq 0.5 \times |c|$. In this case, the centroid has to be refreshed and the alignment is computed again for all sequences of this cluster.

4.1.2. *Comparing Sequences and Centroids.* Let s be the current sequence and C the set of all clusters. Our algorithm scans C and, for each cluster $c \in C$, performs a comparison between s and ς_c (the centroid of c, which is an aligned sequence). This comparison is based on the longest common sub-sequence (LCS) between s and ς_c (see Definition 3). The length of the sequence is also taken into account since it has to be no more than 120% and no less than 80% of the original sequence (*i.e.* the first sequence inserted into c).

Definition 3. *Let s_1 and s_2 be two sequences. Let $LCS(s_1, s_2)$ be the length of the longest common subsequences between s_1 and s_2. The* similarity $sim(s_1, s_2)$ *between s_1 and s_2 is defined as follows:*

$$sim(s_1, s_2) = 1 - \frac{2 \times LCS(s_1, s_2)}{|s_1| + |s_2|}$$

The dissimilarity $d(s_1, s_2)$ *between s_1 and s_2 is defined as follows:*

$$d(s_1, s_2) = 1 - sim(s_1, s_2)$$

Let t be the length of the first sequence inserted into c, s is inserted into c if the three following conditions hold:

- $\forall d \in C / d \neq c, sim(s, \varsigma_d) \leq sim(s, \varsigma_c)$;
- $0.8 \times t \leq |s| \leq 1.2 \times t$;
- $sim(s, \varsigma_c) \geq 0.7$.

The first condition ensures that s will be inserted into a cluster having the most similar centroid to s. The second condition ensures that after inserting s, c will contain sequences of similar length and that the clusters' average length will not vary too much. Finally, the third condition ensures that ς_c and s are similar (with a degree of 70%). If there is no cluster such that these conditions hold, then a new cluster is created and s is inserted into this new cluster.

Algorithm Data Preparation
Input: Output:

(1) Foreach request, parse the parameters and build the corresponding objects.
(2) Build M, the similarity matrix between each pair of objects ;
(3) $\forall p \in M, Neighbors_p \leftarrow$ sorted list of neighbors for p (the first object in the list of p is the closest to p).
(4) $DensityList \leftarrow$ sorted list of objects by density ;

End algorithm Data Preparation

FIGURE 3. Algorithm Data Preparation

4.2. **Clustering PHP requests.** In this section we present the preprocessing principle we use to build objects from the PHP requests, the similarity we propose between them and our clustering algorithm.

Preprocessing. We focus on PHP scripts since there are still potential malicious usage of these scripts and an algorithm intending to extract atypical usage behaviours might help identifying attacks. For each batch of the usage data stream, our preprocessing step firstly filters the data corresponding to PHP scripts, with parameters. Secondly, for each parameter, we build an object corresponding to the keyword given by the user. Let us consider, for instance, the following request: `staff.php?FName=John&LName=Doe`. The corresponding objects are $o_1 =$`John` and $o_2 =$`Doe`. Therefore, at the end of this preprocessing step, we are provided with the set of all the parameters extracted from the batch.

Similarity Between Objects. We consider each object as a sequence of characters. Our comparison of two parameters is then based on the dissimilarity between two words, based on the LCS as described in definition 3.

Example 2. *Let us consider two parameters* p_1=`intrusion` *and* p_2=`induction`*. The LCS between* p_1 *and* p_2 *is* L=`inuion`*.* L *has length 6 and the dissimilarity between* p_1 *and* p_2 *is* $d = 1 - \frac{2\times L}{|p_1|+|p_2|} = 33.33\%$*. Which also means a similarity of* 66.66% *between both parameters.*

The goal of this paper is to focus on our new outlier detection paradigm (described in Section 5). Therefore, the clustering algorithm we propose has to be well adapted to data streams and PHP requests (though it is not the center of our contribution). Actually, we could have used a k-means algorithms as well as hierachical or neural methods, for instance. The main interest of this clustering step is to show the impact of a distribution variation on the outlier detection. Such a variation may occur with any clustering algorithm and any distance or similarity measure. This will be discussed in the experiments. Algorithm **Clustering** (Figure 4) is based on an agglomerative principle. The principle of our clustering algorithm is to increase the volume of clusters by adding candidate objects, until the Maximum Dissimilarity (MD) is broken (*i.e.* there is one object o_i in the cluster such that the dissimilarity between o_i and the candidate object o_c is greater than MD). Our preprocessing step is illustrated by Algorithm **Data Preparation** (Figure 3).

5. PARAMETERLESS OUTLIER DETECTION

Most previous work in outlier detection requires a parameter [15, 32, 25, 16], such as a percent of small clusters that should be considered as outliers or the top-n outliers.

Algorithm Clustering
Input: U, the objects
and MD, the Maximum Dissimilarity.
Output: C, the set of as large clusters as possible,
respecting MD.

(1) $i \leftarrow 0$; $C \leftarrow \emptyset$;
(2) $p \leftarrow$ next unclassified object in $DensityList$;
(3) $i++$; $c_i \leftarrow p$;
(4) $C \leftarrow C + c_i$;
(5) $q \leftarrow$ next unclassified object in $Neighbors_p$;
(6) $\forall o \in c_i$ If $d(o,q) > MD$ then return to step 2 ;
(7) add q to c_i ;
(8) return to step 5 ;
(9) If unclassified objects remain then return to step 2 ;
(10) return C ;

End algorithm Clustering

FIGURE 4. Algorithm Clustering

Generally, their key idea is to sort the clusters by size and/or tightness. We consider that our clusters will be as tight as possible, according to our clustering algorithm, and we aim to extract outliers by sorting the clusters by size. The problem is to separate "big" and "small" clusters without any *apriori* knowledge about what is big or small. Our solution is based on an analysis of cluster distribution, once they are sorted by size. One possible distribution shape is illustrated in figure 5 (screenshot made with our real data). The key idea of WOD is to use a wavelet transform to cut down such a distribution. With a prior knowledge on the number of plateaux (we want two plateaux, the first one standing for small groups, or outliers, and the second one standing for big groups, or clusters) we can cut the distribution in a very effective manner. In figure 5, the y axis stands for the size of the clusters, whereas their index in the sorted list is represented on x, and the two plateaux allow separating small and big clusters. Actually, each cluster having size lower than (or equal to) the first plateau will be considered as an outlier.

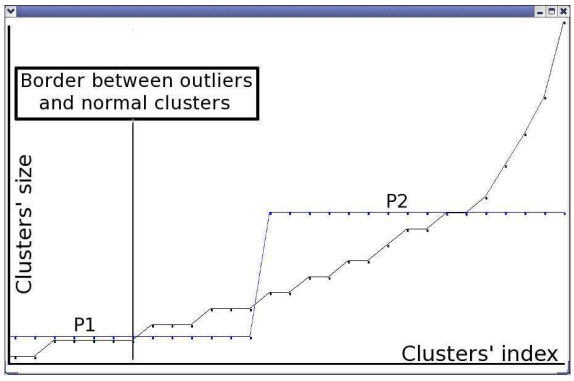

FIGURE 5. Detection of outliers by means of Haar Wavelets

The wavelet transform is a tool that cuts up data or functions or operators into different frequency components, and then studies each component with a resolution matched to its scale [8, 7]. In other words, wavelet theory represents series of values by breaking them down into many interrelated component pieces; when the pieces are scaled and translated wavelets, this breaking down process is termed wavelet decomposition or wavelet transform. Wavelet reconstructions or inverse wavelet transforms involve putting the wavelet pieces back together to retrieve the original object [30]. Mathematically, the continuous wavelet transform is defined by:

$$T^{wav} f(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{+\infty} f(x)\psi^*(\frac{x-b}{a})dx$$

where z^* denotes the complex conjugate of z, $\psi^*(x)$ is the analyzing wavelet, a (> 0) is the scale parameter and b is the translation parameter. This transform is a linear transformation and it is co-variant under translations and dilations. This expression can be equally interpreted as a signal projection on a function family analyzing $\psi_{a,b}$ constructed from a mother function in accordance with the following equation: $\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi(\frac{t-b}{a})$. Wavelets are a family of basis functions that are localized in time and frequency and are obtained by translations and dilations from a single function $\psi(t)$, called the mother wavelet. For some very special choices of a, b, and ψ, $\psi_{a,b}$ is an orthonormal basis for $L^2(\mathbb{R})$. Any signal can be decomposed by projecting it on the corresponding wavelet basis function. To understand the mechanism of wavelet transform, we must understand the multiresolution analysis (MRA). A multiresolution analysis of the space $L^2(\mathbb{R})$ consists of a sequence of nested subspaces such as:
$... \subset V_2 \subset V_1 \subset V_0 \subset V_{-1} ... \subset V_{j+1} \subset V_j ...$

$$\overline{\bigcup_{j\in Z} V_j} = L^2(\mathbb{R})$$

$$\bigcap_{j\in Z} V_j = \{0\}$$

$\forall j \in \mathbb{Z}$ if $f(x) \in V_j \iff f(2^{-1}x) \in V_{j+1}$
(or $f(2^j x) \in V_0$)
$\forall k \in \mathbb{Z}$ if $f(x) \in V_0 \iff f(x-k) \in V_0$

There is a function $\varphi(x) \in L^2(\mathbb{R})$, called scaling function, which by dilation and translation generates an orthonormal basis of V_j. Basis functions are constructed according to the following relation :

$\varphi_{j,n}(x) = 2^{-\frac{j}{2}}\varphi(2^{-j}x - n), n \in \mathbb{Z}$, and the basis is orthonormated if $\int_{-\infty}^{+\infty} \varphi(x)\varphi^*(x+n)dx = \delta(n), n \in \mathbb{Z}$. For each V_j, its orthogonal complement W_j in V_{j-1} can be defined as follows:
$V_{j-1} = V_j \oplus W_j$ and $L^2(\mathbb{R}) = \bigoplus_{j\in\mathbb{Z}} W_j$. As W_j is orthogonal to V_{j-1}, then W_{j-1} is orthogonal to W_j, so $\forall j, k \neq j$ then $W_j \perp W_k$.

There is a function $\psi(x) \in \mathbb{R}$, called wavelet, which by dilations and translations generates an orthonormal basis of W_j, and so of $L^2(\mathbb{R})$. The basis functions are constructed as follows:

$$\psi_{j,n}(x) = 2^{-\frac{j}{2}}\psi(2^{-j}x - n), n \in \mathbb{Z}$$

Therefore, $L^2(\mathbb{R})$ is decomposed as a direct sum of the spaces W_j which becomes an orthogonal sum [7], i.e. $L^2(\mathbb{R}) = \bigoplus_{j\in Z} W_j$. To summarize the wavelet decomposition: given a f_n function in V_n, f_n is decomposed into two parts, one part in V_{n-1} and the

other in W_{n-1}. At next step, the part in V_{n-1} continues to be decomposed into two parts, one part in V_{n-2} and the other in W_{n-2} and so on. Figure 6 gives an illustration of the multiresolution analysis.

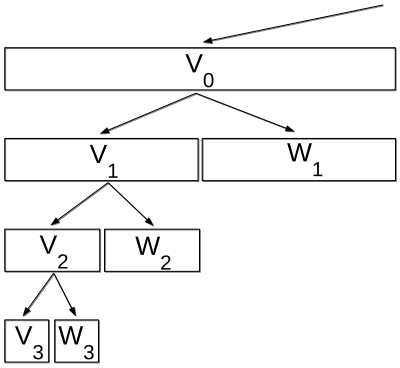

FIGURE 6. Multiresolution Analysis Principle

A direct application of multiresolution analysis is the fast discrete wavelet transform algorithm. The idea is to iteratively smooth data and keep the details all along the way. More formal proofs about wavelets can be found in [8, 30]. The wavelet transform provides a tool for time-frequency localization and are generally used to summarize data and to capture the trend in numerical functions. In practice, the majority of wavelets coefficients are small or insignificant, so to capture the trend only a few significant coefficients are needed. We use the Haar wavelets to illustrate our outlier detection method. Let us consider the following series of values: $[1, 1, 2, 5, 9, 10, 13, 15]$. Its Haar wavelet transform is illustrated by table 7.

Level	Approximations	Coefficients
8	1, 1, 2, 5, 9, 10, 13, 15	
4	1, 3.5, 9.5, 14	0, -1.5, -0.5, -1
2	2.25, 11.75	-1.25, -2.25
1	7	-4.75

FIGURE 7. Wavelet transform of $[1, 1, 2, 5, 9, 10, 13, 15]$

Then, we keep only the two most significant coefficients and we make the others zero. In our series of coefficients ($[7, -4.75, -1.25, -2.25, 0, -1.5, -0.5, -1]$) the most two significant ones are 7 and -4.75, meaning that the series becomes $[7, -4.75, 0, 0, 0, 0, 0, 0]$. In the following step, the inverse operation is calculated and we obtain an approximation of the original data $[2.25, 2.25, 2.25, 2.25, 11.75, 11.75, 11.75, 11.75]$. This gives us two plateaux corresponding to values $\{1, 1, 2, 5\}$ and $\{9, 10, 13, 15\}$. The set of outliers contains all the clusters having size smaller than the first plateau(*e.g.* 2.25). In our example, $o = \{1, 1, 2\}$ is the set of outliers.

More generally, advantages of this method, for our problem, are illustrated in figures 8 and 9. Depending on the distribution, wavelets will give different indexes (where to cut). For instance, with few clusters having the maximum size (see graph with solid lines from

figure 8), wavelets will cut the distribution in the middle. On the other hand, with a large number of such large clusters (see graph with dashed lines from figure 8), wavelets will accordingly increase the number of clusters in the little plateau (taking into account the large number of big clusters). Furthermore, in our usage data at INRIA, there is a variation between night and day in the usage of PHP scripts. This variation results into two main shapes. Figure 9 gives an illustration of two different distributions, similar to the ones we found out in our experiments. Let us consider the 10% filter on this distribution, which aims to isolate outliers corresponding to 10% of the global shape. If one uses the 10% percent filter in order to detect outliers, one will obtain a relevant outlier detection for the first distribution (corresponding to usage of scripts at 1 am). However, with the second distribution (calculated from the usages at 5 pm), this filter will give a very high value and return clusters that should not be considered as outliers. On the other hand, our wavelet based filter will adjust to the distribution variation and the threshold for outlier detection will only slightly increase, taking into account the new distribution shape.

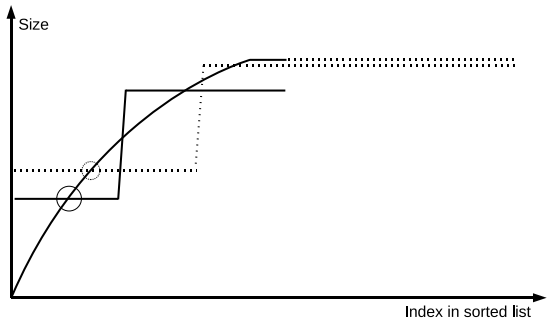

FIGURE 8. Automatic adjustment to the clustering results

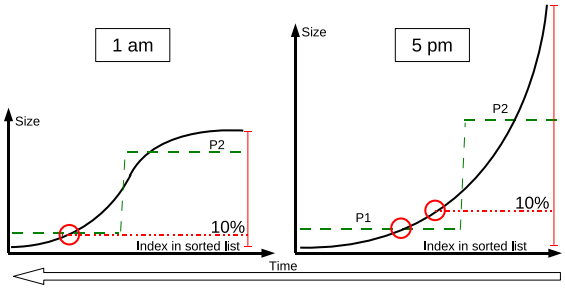

FIGURE 9. A Distribution Varying With Time

Applying the wavelet transform on the series allows us to obtain a good data compression and, meanwhile, according to different trends, a good separation. Knowing that outliers are infrequent objects, they will always be grouped into small clusters. WOD's principle of separating outliers from clusters is based on theorem 1.

Theorem 1. *Let F be the set of features used to build the distribution D on the clustering result. Let P1 and P2 be the two plateaux obtained after selecting the most two significant*

coefficients of the wavelet transform on D. The optimal separation into two groups according to F and regarding the minimisation of the sum squared error, is given by P1 and P2.

Proof In an orthonormal base, it has been shown that keeping the largest k wavelet coefficients gives the best k-term Haar approximation to the original signal, in terms of minimizing the sum squared error for a given k [29]. For this propose, let us consider the original signal $f(x)$ and the basis functions $u_1(x)$, ...$u_m(x)$. The signal can thus be represented depending on the basis functions as :

$$f(x) = \sum_{i=1}^{m} c_i u_i(x)$$

The goal is to find an approximating function with fewer coefficients. Let σ be a permutation of $1, ..., m$ and f' the approximating function using only the first m' elements of σ, with $m' < m$.

$$f'(x) = \sum_{i=1}^{m'} c_{\sigma(i)} u_{\sigma(i)}(x)$$

The square of L^2 error of this approximation is:

$$\begin{aligned} ||f(x) - f'(x)||_2^2 &=< f(x) - f'(x) | f(x) - f'(x) > \\ &= \left\langle \sum_{i=m'+1}^{m} c_{\sigma(i)} u_{\sigma(i)} | \sum_{j=m'+1}^{m} c_{\sigma(j)} u_{\sigma(j)} \right\rangle \\ &= \sum_{i=m'+1}^{m} \sum_{j=m'+1}^{m} c_{\sigma(i)} c_{\sigma(j)} < u_{\sigma(i)} | u_{\sigma(j)} > \\ &= \sum_{i=m'+1}^{m} (c_{\sigma(i)})^2 \end{aligned}$$

Due to the basis orthonormality, $< u_i, u_j >= \delta$, so, for any $m' < m$, to minimize this error the best choice for σ is the increasing permutation (or the permutation that contains the elements ordered in increasing order).

Therefore, for $m' = 2$ we obtain the best 2-term Haar approximation to the original signal.□

Based on theorem 1, we select the clusters having size smaller than the first plateau. These clusters can be considered as outliers without any parameter given by the end-user.

6. EXPERIMENTS

The goal of our experiments is to show the advantages of our parameterless outlier detection in a streaming environment. In such an environment, choosing a good level of outlyingness is highly difficult given the short time available to take a decision. In this context, an outlier detection method which does not depend on a parameter such as k, for the top-k outliers, or a percentage p of small clusters, should be much appreciated. On the other hand, such a parameterless outlier detection method also has to guarantee good results. This method should be able to provide the end-user with an accurate separation into small and big clusters. It should also be able to fit any kind of distribution shape

(exponential, logarithmic, linear, etc.). Finally, it should also be able to automatically adjust to the number of clusters and to their size from one batch to the other. Our claim is that WOD matches all these requirements and we illustrate these features in this section . For these experiments we used real data, coming from the Web Log usage of INRIA Sophia-Antipolis from January 2006 to April 2007. The original files have a total size of 18 Gb.

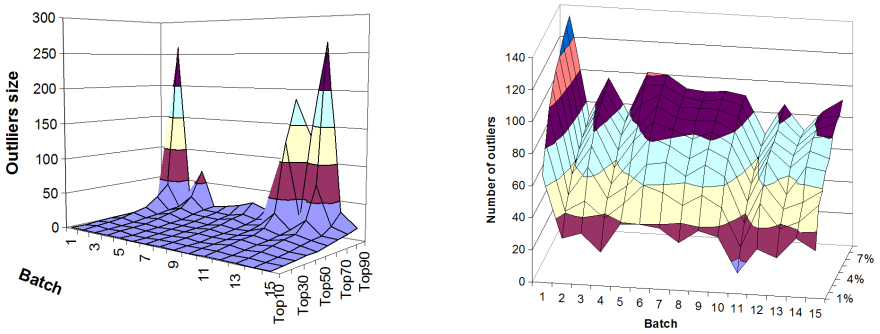

FIGURE 10. size of outliers with a top-k filter and number of outliers with a $\%$ filter on dataset *navigations*

6.1. **Navigations.** The access log files of this experiments correspond to a total of 11 millions navigations that have been split into batches of 8500 requests each (in average). In this section, we report some results on the first 15 batches, since they are very representative of the global results.

Figure 10 shows the behaviour of two filters on the first 15 batches. Foreach batch, the number of objects (navigation sequences) and clusters is given in Table 1. The first filter (left part of Figure 10) shows the size of clusters selected by a top-k filter. The principle of this filter is to select only the first k clusters after sorting them by size. An obvious disadvantage of this filter is to select either too much or not enough clusters. Let us consider, for instance, batch 13 in this figure. With $k = 50$ the maximum outliers size is 12, whereas with $k = 90$ this size is 265 (which is the maximum size of a cluster in this batch since it contains only 87 clusters). Another disadvantage is to arbitrary select or ignore clusters with equal size. For instance, with $s = \{1, 1, 2, 2, 3, 5, 10\}$ a series of sizes and $k = 3$, the top-k filter will select the 3 first clusters having sizes: 1, 1 and 2, but will ignore the 4^{th} cluster having size 2.

We also have implemented a filter based on p, a percentage of clusters, to select outliers. The number of outliers selected by this filter with different values of p (*i.e.* from 0.01 to 0.09) are given in Figure 10 (right). The principle is to consider $p \in [0..1]$, a percentage given by the end-user, $d = maxVal - minVal$ the range of cluster sizes and $y = (p \times d) + minVal$. Then, the filter aims to select only clusters having size s, such that $s \leq y$. For instance, with $s = \{1, 3, 10, 11, 15, 20, 55, 100\}$ a series of sizes and $x = 0.1$ we have $d = 100 - 1 = 99$, $y = 1 + (0.1 \times 99) = 10$ and the set of outliers will be $o = \{1, 3, 10\}$. In our experiments, this filter is generally better than a top-k filter. Actually, we can notice homogeneous results from Figure 10. For instance, with batch 13 we can see a number of outliers ranging from 24 (1 %) to 70 (9 %).

Batch	1	2	3	4	5	6	7	8
Number of Objects	1391	2076	2234	1635	2174	1672	1955	2009
Number of Clusters	154	92	118	98	128	116	119	111
Batch	9	10	11	12	13	14	15	
Number of Objects	2455	2182	2857	2498	2294	2698	2090	
Number of Clusters	119	108	82	94	87	106	122	

TABLE 1. Batches, objects and clusters

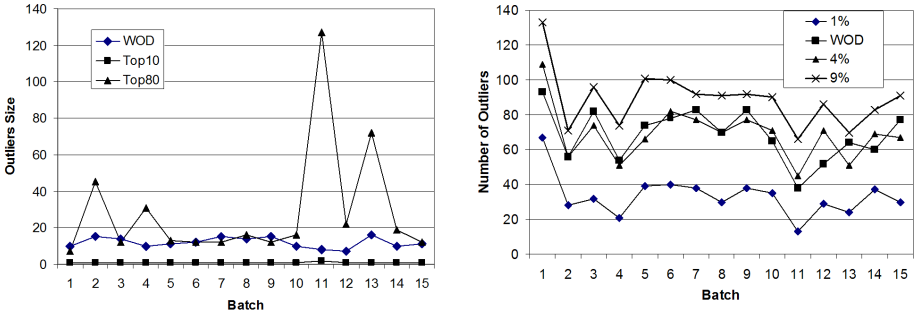

FIGURE 11. Comparing WOD with top-k and $\%$ filters on dataset *navigations*

Figure 11 gives a comparison of WOD (applied to the same data) with top-k and percentage filtering. The left graph reports the results when comparing WOD with a top-10 and a top-80 filter. Filter top-80 gives good results for approximately 50% of the batches, whereas filter-10 always gives very low values (size 1 or 2). Unfortunately, a value of 80 for this filter cannot be considered as a reference. For instance, in batch number 11, we notice a cluster having size 127 is considered an outlier. The maximum size of a cluster in batch 11 is 172 and there are 82 clusters. This result is thus not acceptable and shows that top-k is unable to adjust to changes in the distribution of cluster sizes. On the other hand, thanks to its wavelet feature, WOD is able to automatically adjust and will select 8 as a maximum size of an outlier.

On the right graph of Figure 11, we focus on three percentage filters with 1%, 4%, 9% and we compare them to WOD. For instance, with batch 11, we know that WOD labels clusters having size less than or equal to 8 as outliers. That filtering gives a total of 38 clusters (where filter 4 % gives 45 outliers). These clusters represent a total of 184 objects in a batch which contains 2294 objects. Our observation is that WOD and 4% would give similar results. However, we discuss the advantages of WOD on the percentage filter in subsection 6.3.

6.2. **PHP parameters.** For these experiments, the log files have been split into batches of 10,000 php requests. We focus on 16 batches, since they are representative of the global results and they illustrate the variation of distribution. The first 8 batches have been selected among PHP request occuring between 1 and 2 am, and the 8 former have been selected among requests occuring between 3 and 4 pm. Figure 12 shows the behaviour of filters top-k and p% on those 16 batches. First surface in Figure 12 (left) shows the size of clusters selected by a top-k filter. Let us consider, for instance, batch 13 in Figure 12. With

$k = 50$ the maximum outliers size is 4, whereas with $k = 90$ this size is 67 (which is the maximum size of a cluster in this batch, which contains only 84 clusters).

Regarding the percentage filter, the number of outliers selected by this filter with different values of p (*i.e.* from 0.01 to 0.09) is given in figure 12 (right surface). Once again, we can notice homogeneous results for this filter in Figure 12. For instance, with batch 13 we can see a number of outliers ranging from 44 (1 %) to 78 (9 %), which corresponds to the results of top-40 to top-70.

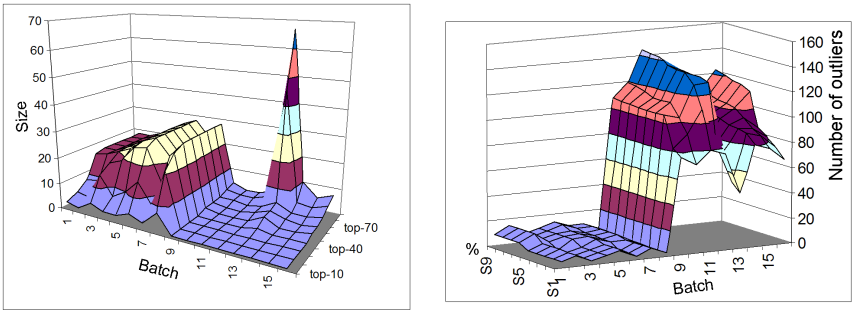

FIGURE 12. size of outliers with a top-k filter and number of outliers with a p% filter on dataset *PHP*

Figure 13 gives a comparison of WOD (applied to the same data) with top-k and percentage filtering. In the left part of Figure 13, we compare WOD with a top-10 and a top-70 filter. For the first 8 batches, top-10 and WOD give the best result. Unfortunately, for batches 9 to 16, top-10 returns too few outliers (having maximum size 1). Therefore, this filter cannot be used for the whole stream. The best top-k results for batches 9 to 16 are given by a top-70. Unfortunately, its results for batches 1 to 8 are bad (values are too high, with outlier having size up to 28). Therefore, no value of k in this filter can be considered as a reference. The end-user would have to modify the value of k from one batch to another. This result is thus not acceptable and shows that top-k is unable to adjust to changes in the distribution of cluster sizes. On the other hand, thanks to its wavelet feature, WOD is able to automatically adjust and will always select a correct maximum size to detect atypical events.

In the right part of Figure 13, we focus on two percentage filters (*i.e.* 1% and 5%) and we compare them to WOD. Our observation is that WOD and 1% would give similar results. For instance, with batch 7, we know that WOD labels clusters having size less than or equal to 3 as outliers. That filtering gives a total of 9 clusters (where filter 1 % gives 8 outliers). We also can observe that most of the values given by filter 1 % on the first 8 batches are low. Filter 5 % gives better values for the first 8 batches (very similar to WOD) but it has bad results on the next 8 batches (up to 138 outliers). This is due to the variation of the distribution, as illustrated by Figure 9.

6.3. **Discussion.** These experiments illustrate the advantages of WOD over the traditional filters:

(1) WOD does not require any parameter tuning. It adjusts automatically, whatever the distribution shape and the number of clusters. In contrast, the end-user will have to try several percentage values before finding the good range (*i.e.* between

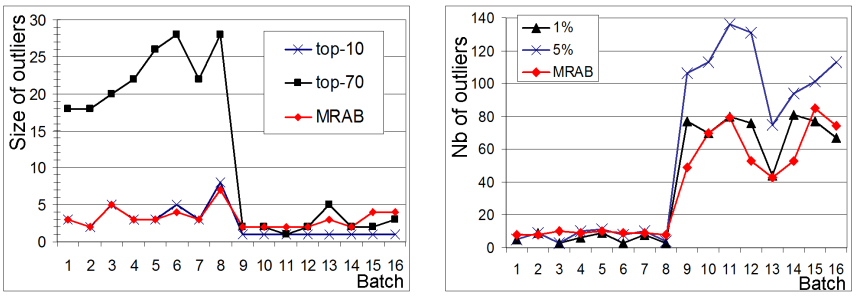

FIGURE 13. Comparison with WOD for top-k and p%

3% and 9% the percentage filter gives good outliers for the first batches). Furthermore, the outlier detection provided by WOD will not degrade with a variation of distribution shape over time. In our case, the distribution is usually exponential. Let us consider a change of usage, or a change of clustering method, resulting in a variation of the distribution shape. That new shape could be logarithmic, for instance, such as the distribution illustrated in Figure 8. Then, the percentage filter would have to be manually modified to fit that new distribution, whereas WOD would keep giving the good set of outliers without manual settings.

(2) WOD gives a natural separation between small and big values (according to theorem 1). Let us consider our previous illustration of a distribution $s = \{1, 3, 10, 11, 15, 20, 55, 100\}$. We know that on this distribution a 10% filter would give the following set of outliers : $o = \{1, 3, 10\}$. However, why not including 11 into o? Actually, 10 and 11 are very close values. On the other hand, with WOD we have $o = \{1, 3\}$, which is obviously a natural and realistic result.

7. CONCLUSION

In this paper we have presented WOD, an outlier detection method that does not require any manual tuning. Our principle is first based on a distribution of clusters according to some characteristics such as their size, tightness, density or any other characteristic. Thanks to its wavelet feature, WOD is able to cut down this distribution into two sets, corresponding to clusters and outliers. The advantages of WOD are i) automatic adjustment to distribution shape variations and ii) relevant and accurate detection of outliers with very natural results. Our experiments, performed on real data, confirm this separation feature of WOD compared to well-known outlier detection principles such as the top-k outliers or the percentage filter.

REFERENCES

[1] C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. *SIGMOD Records*, 30(2):37–46, 2001.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD Conf.*, pages 207–216, Washington DC, USA, May 1993.

[3] E. Aleskerov, B.and Freisleben, and B. Rao. Cardwatch: A neural network based database mining system for credit card fraud detection. In *IEEE Computational Intelligence for Financial Engineering*, 1997.

[4] N. Billor, A. S. Hadi, and P. F. Velleman. BACON: blocked adaptive computationally efficient outlier nominators. *Computational Statistics and Data Analysis*, 34, 2000.

[5] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. *SIGMOD Records*, 29(2):93–104, 2000.
[6] W. Chimphlee, A. H. Abdullah, M. N. Md Sap, and S. Chimphlee. Unsupervised anomaly detection with unlabeled data using clustering. In *International conference on information and communication technology*, 2005.
[7] Charles K. Chui. *An introduction to wavelets*. Academic Press Professional, Inc., San Diego, CA, USA, 1992.
[8] Ingrid Daubechies. *Ten lectures on wavelets*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.
[9] L. Ertoz, E. Eilertson, A. Lazarevic, P.-N. Tan, V. Kumar, J. Srivastava, and P. Dokas. Minds - minnesota intrusion detection system. *Data Mining - Next Generation Challenges and Future Directions*, 2004.
[10] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. *Applications of Data Mining in Computer Security*, 2002.
[11] H. Fan, O. R. Zaiane, A. Foss, and J. Wu. A nonparametric outlier detection for effectively discovering top-n outliers from engineering data. In *PAKDD*, 2006.
[12] R. Fujimaki, T. Yairi, and K. Machida. An approach to spacecraft anomaly detection problem using kernel feature space. In *11th ACM SIGKDD*, 2005.
[13] Z. He, X. Xu, and S. Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24, 2003.
[14] M. F. Jaing, S. S. Tseng, and C. M. Su. Two-phase clustering process for outliers detection. *Pattern Recogn. Lett.*, 22(6-7):691–700, 2001.
[15] W. Jin, A. K. H. Tung, and J. Han. Mining top-n local outliers in large databases. In *7th ACM SIGKDD*, pages 293–298, 2001.
[16] J. Joshua Oldmeadow, S. Ravinutala, and C. Leckie. Adaptive clustering for network intrusion detection. In *8th PAKDD*, volume 3056 of *Lecture Notes in Computer Science*, pages 255–259, 2004.
[17] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *24rd International Conference on Very Large Data Bases*, pages 392–403, 1998.
[18] H. Kum, J. Pei, W. Wang, and D. Duncan. ApproxMAP: Approximate mining of consensus sequential patterns. In *Proceedings of SIAM Int. Conf. on Data Mining*, San Francisco, CA, 2003.
[19] R. Kwitt and U. Hofmann. Unsupervised anomaly detection in network traffic by means of robust pca. In *International Multi-Conference on Computing in the Global Information Technology*, 2007.
[20] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *IEEE Symposium on Security and Privacy*, 2001.
[21] Tao Li, Qi Li, Shenghuo Zhu, and Mitsunori Ogihara. A survey on wavelet applications in data mining. *SIGKDD Explor. Newsl.*, 4(2):49–68, 2002.
[22] Alice Marascu and Florent Masseglia. Mining sequential patterns from data streams: a centroid approach. *J. Intell. Inf. Syst.*, 27(3):291–307, 2006.
[23] M. Markou and S. Singh. Novelty detection: a review - part 1: statistical approaches. *Signal Processing*, 83, 2003.
[24] S. Papadimitriou, H. Kitagawa, P.B. Gibbons, and C. Faloutsos. LOCI: fast outlier detection using the local correlation integral. In *19th International Conference on Data Engineering*, 2003.
[25] L. Portnoy, E. Eskin, and S. Stolfo. Intrusion detection with unlabeled data using clustering. In *ACM CSS Workshop on Data Mining Applied to Security*, 2001.
[26] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. *SIGMOD Records*, 29(2):427–438, 2000.
[27] Karlton Sequeira and Mohammed Zaki. Admit: anomaly-based data mining for intrusions. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 386–395, New York, NY, USA, 2002. ACM.
[28] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 428–439, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
[29] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. Wavelets for computer graphics: A primer, part 1. *IEEE Computer Graphics and Applications*, 15(3):76–84, 1995.
[30] Randy K. Young. *Wavelet Theory and Its Applications*. Kluwer Academic Publishers Group, 1995.
[31] Dantong Yu, Gholamhosein Sheikholeslami, and Aidong Zhang. Findout: finding outliers in very large datasets. *Knowl. Inf. Syst.*, 4(4):387–412, 2002.
[32] S. Zhong, T. M. Khoshgoftaar, and N. Seliya. Clustering-based network intrusion detection. *International Journal of Reliability, Quality and Safety Engineering*, 14, 2007.