Adaptive Threshold for Outlier Detection on Data Streams

James P. Clark
Centralized Super Computing Facility
Lockheed Martin
Herndon, V.A., USA
james.p.clark@lmco.com

Zhen Liu
School of Medical Inofrmation Engineering
Guangdon Pharmaceutical University
Guangzhou, China
liu.zhen@gdpu.edu.cn

Nathalie Japkowicz
Computer Science Department
American University
Washington D.C., USA
japkowic@american.edu

***Abstract*—As the distribution of a data stream evolves over time, a learner must adapt to its distributional shifts in order to make accurate predictions. In the context of anomaly detection, it is crucial for the learner to distinguish between natural changes in distribution and true anomalies in the data stream. This is the problem we focus on in this study which considers the situation where only normal data are available for initial training, but subsequent data can be either normal or anomalous. In that context, it is necessary to train a one-class learning anomaly detection system on the normal data and let the system output a score representing the degree of normalcy or outlierness that each data point in the subsequent data stream exhibits. The system then uses a threshold to discriminate between normal and anomalous instances. In the case of data stream, the data distribution may shift overtime, and a fixed threshold could develop a high false alarm rate or a low outlier detection rate in case of concept drift. To this end, we designed an adaptive sliding window approach which updates the threshold when necessary based on the scores distribution. Experimental results show that our method improves the performance of base anomaly detectors by dynamically updating the threshold of the scores when needed rather than using a fixed threshold or an adaptive threshold with fixed window sizes.**

***Index Terms*—outlier detection, threshold setting, one class learning, auto encoder, LOF**

I. Introduction

The task of process monitoring or outlier detection is becoming increasingly important in modern applications. For example, in the medical realm, continuously monitoring a patient's vital signs and alerting an emergency center in cases where a deterioration of the patients condition is detected can save lives. In the cyber security domain as well, a network can be continuously monitored and an alert issued in the event of a cyber attack. This could prevent disastrous situations from developing. Recently, process monitoring in continuously incoming data streams has been proposed using an extension of a classical one-class learning system [9]. In that work, as well as in other one-class learning work applied to batch or streaming data (e.g. [14], [15], etc.), the learning process is divided into two parts: concept learning and threshold-setting. The threshold is used to separate the scores output by the model into outliers and normal data. It has often been assumed that a threshold can be set manually by the user through inspection of the inductive learning algorithms results. However, in non-stationary environments, the threshold may change over-time making it undesirable for a user to set it once at the beginning of the process deployment and let it run indefinitely. In order to adapt a batch one-class learning outlier detector to the streaming data case, therefore, it is necessary to develop an adaptive threshold setting method able to meet the following requirements:

- The method performs well despite a limited and imperfect amount of training data: only an initial batch of normal data is available. The subsequent data is unlabelled and contains both normal and outlier data.
- The method focuses on keeping a low false alarm rate throughout concept drifts so as not to disrupt the functionality of the overall outlier detection system.
- The method must be efficient so as not to disrupt the operation of the anomaly detector on the very fast arriving stream of data.

These are difficult challenges given that, at the best of times, setting a threshold is a challenging task since the boundary between normal data and outliers is, in most cases, unclear. Furthermore, one-class learning systems must perform this task without any prior knowledge of outliers.

The purpose of this paper is to present a technique for addressing this problem and comparing its performance to those of simpler approaches that could be designed for this problem. In more detail, our focus is on the task of adaptive threshold setting in the case of outlier detection in data streams with no outliers available for training. We chose this particular setting because of its practical utility. To illustrate this last point, consider, for example, the case of a patient being fitted with a monitoring system designed to alert a medical center in case of a medical emergency. The system is fitted while the patient's condition is stable. Therefore, the only data available at deployment time is the data representing his or her stable state (i.e., normal data). The goal of the system is to issue an alarm when the patient's state deviates too much from its normal state while keeping in mind that the patient's normal state may evolve (concept drifts) and that the system must be able to discriminate between a benign evolution (new normal data) and a true emergency (outlier). This adaptation to new conditions needs to happen based solely on the unlabelled data that streams through the system after deployment.

It is clear that the solution of setting a fixed threshold at deployment time is not sufficient as this threshold needs to evolve in order for the system not to issue huge numbers of false alarms. A second solution is to use fixed-size sliding windows and adjust the threshold based on the performance of the outlier detection system in these windows. This solution does allow the threshold to adapt, but it does not allow the window size to change as necessary. As a result the solution is not flexible enough. Indeed, while a fixed-size small window would yield good anomaly detection performance, it would be inefficient since the threshold would have to be reset very frequently and often unnecessarily. A large fixed-size window, on the other hand, would be more efficient but would probably miss some context drifts and cause large numbers of false alarms. Our solution, in contrast, uses sliding windows but allows them to vary in size as necessary so as to achieve the good performance of small windows while avoiding their inefficiency. A related and important feature of our approach is that it avoids the difficult task necessary in the fixed-size window approach of setting the window size. Instead, that determination is done automatically and automatically revised whenever it is needed.

In more detail, our method uses one class learning algorithms as base algorithms for outlier detection and considers the scores such methods output when presented with new instances. The method applies a statistical test to detect significant changes to the mean of these scores. Significant changes in mean scores suggest that the output of the outlier detection algorithm changed and that the threshold needs an update in order to function in the new environment. No significant change means that the system can continue proceeding as is. Depending on the frequency with which a significant change is detected, the size of the window used to recalculate the threshold will be different. The entire process is unsupervised and has a very low time cost since the cost of running a statistical test is much lower than that of recalculating a threshold. The approach was compared to the static threshold method and the method that consists of using a fixed-size window (two sizes were considered). Our experiments used two one-class learning algorithms (Autoencode, LOF and HS-Tree) as base anomaly detectors and were carried out on four artificial data sets and two real-world data sets. The experimental results show that our method is superior to its competitors on the task of outlier detection in data streams.

The remainder of the paper is divided into 5 sections. Section 2 discusses related work in this area of research. Section 3 presents our method. Section 4 and 5 describe our experimental setup and results, respectively. Finally, Section 6 concludes the paper and suggests avenues for future work.

II. Related Works

Outlier detection methods fall mostly into the following three categories:

- Distance based outlier detection ([1], [2]);
- Density based outlier detection ([3]);
- Clustering based outlier detection ([4]).

In the distance-based paradigm, a data point d is considered an outlier if fewer than k data points are located at a distance R from d, or the distance from d to its k-nearest neighbors exceeds a given threshold. In the density-based paradigm, outliers are the data points that are surrounded by a density dissimilar to that of their local neighbors. The local reachability density is used to score the degree of outlierness of each data point. Finally, in the clustering-based paradigm, methods group similar data points into a cluster, and the outliers are identified as the data points that lie far away from the center of their nearest clusters.

One-class learning methods have also been used to detect outliers (e.g., [5]). Such methods share similarities with the outlier detection approaches just discussed but are used in a different context: while traditional outlier detection methods are applied to data sets in an attempt to identify the outliers present in these data sets, one-class learning approaches are more dynamic in nature: they are trained on data sets devoid of outliers and applied to subsequently arriving data to assess whether they represent a normal instance or an outlier. They proceed by outputting a score, based on their preliminary training, that reflects the degree of outlierness of previously unknown data points. The same paradigms as those previously discussed can be implemented in the context of one-class anomaly detection.

Swersky et al. [10] compared multiple outlier detection methods, including traditional and one-class learning approaches, and concluded that LOF and SVDD outperformed the other paradigms tested. These tests, however, were performed in the static context.

More closely related to this study, previous work already attempted to adapt outlier detection methods to the data streaming context. Each of the paradigms previously discussed were attempted in this context. In [6], a sliding-window is used together with the nearest-neighbors algorithm to detect distance-based outliers in the data stream. New sets of nearest neighbors are calculated in each new window. In [7], the Anyout algorithm aims to solve the outlier detection problem in a data stream using hierarchical clustering. The method is based on clustering and feature weighting. It proceeds by updating the centroid of each cluster together with the weights of the features in each chunk of data. In [8], a density-based method was introduced in the form of an incremental version of the LOF method. It proceeds by incrementally updating the k-nearest neighbors distances, the reachabilty distances, and the local reachability density of the affected points every time a new point is inserted.[1] In [9], an autoencoder is used for outlier detection in data streams. The model is retrained every time a concept drift occurs.

The most important contribution that the works just mentioned made was in deciding what data to consider when adapting an anomaly detection system to the streaming case. They left the important question of how to set the threshold between normal and abnormal data out. The purpose of our

[1] The metrics just introduced will be formally defined in section 3.2.

work is to consider this question. We are not the first ones to do so. At least, in the context of static threshold setting, two kinds of methods have been considered. The Top n kind of method takes the n data points that received the highest scores and marks them as outliers [12]. The statistical approach, on the other hand, considers a point as an outlier if its score is greater than the Mean of the distribution plus α times the Standard Deviation of the distribution, where α is a parameter chosen by the user [13]. There is one case where adaptive threshold setting has been proposed before, but that was done in the case of Time Series [12]. The method is based on the APCA (Adaptive Piecewise Constance Approximation) representation of a time series by segmenting the time series into segments of unequal length based on the data. Unlike in our case, however, this method is suitable for univariate data and not suitable for the multivariate case.

In the remainder of this paper, we describe the method we have designed which outfits a one-class learning anomaly detection system with an adaptive threshold setting method whose aim is to decrease the false alarm rate that a fixed threshold would cause and to avoid the difficulty of choosing the appropriate window size that a fixed-window size method would require.

III. Methodology

This section is divided into two parts. In the first part, we review the one-class learning anomaly detectors that will be used in this study, namely the autoencoder and LOF. In the second part, we present our contribution: the adaptive-window adaptive threshold setting algorithm.

A. Background: One-Class Learning Anomaly Detection Systems

1) Autoencoders: An autoencoder is a feed forward neural network that contains an input layer, an output layer, and one or more hidden layers. The input size and output size are exactly the same. The autoencoders are trained to reconstruct the input at the output layer. An autoencoder can be summed up similarly as a composition of two maps, which we will note as ϕ and ψ, where the goal is to minimize the reconstruction error of inputs X:

$$\phi : X \to F$$

$$\psi : F \to X$$

$$argmin(\phi, \psi)||X - (\phi o \psi)X||^2$$

The reconstruction error is calculated in the mean square error. Autoencoders are trained to minimize this error, which simply means that over time they are trying to output the given input as closely as possible. We define this formally as follows :

$$L(x, w) = L(x, x') = ||xx'||^2 = ||x\sigma_2(w'(\sigma_1(wx+b))+b')||^2$$

.

Based on this equation, the Empirical risk is defined as:

$$E_n(w) = \frac{1}{n} * \sum_{i=1}^{n} L(x_i, w)$$

.

The object of training a model is to minimize the average error through weight adjustment. The stochastic gradient descent optimization procedure along with back propagation are the approaches used throughout this paper.

When identifying a data point, the reconstruction error is taken as a score that expresses the degree of outlierness. If the reconstruction error is small, the point is thought of fitting the model well, and since the model was only trained on normal data, the point is considered normal. If, on the other hand, the reconstruction error is large, then the point is thought not to fit the model and, thus, not to belong to the normal population. It is, thus, considered to be an outlier. The question of what represents a small or a large reconstruction error is exactly the question we ask in this paper since that question can be answered by setting a threshold to discriminate between normal data and outliers.

2) LOF: The local outlier factor is a density based outlier detection method. It evaluates the density around an observation and the density of its nearest neighbors. It is based on the reachability distance defined as:

$$reach - dist_k(p, o) = max(k - distance(o), d(p, o))$$

Where k-distance(o) is the largest distance among the distances to the nearest neighors of *o*. *d*(*p*,*o*) is the distance between p and *o*. The local reachability density of an observation p is the inverse of the average reachability distance based on the k nearest neighbors of p. It is defined as

$$lrd_k(p) = \frac{\sum_{o \in N_k(p)} reach - dist_k(p, o)}{|N_k(p)|}$$

The outlier factor of point p is the average of the ratio of the local reachability density of p and those of p's k nearest neighbors. It calculates the degree of p being an outlier. It is defined as

$$LOF_k(p) = \frac{\sum_{o \in N_k(p)} \frac{lrd_k(o)}{lrd_k(p)}}{|N_k(p)|}$$

The lower p's local reachability density, the higher the local reachability densities of p's k nearest neighbors, and, thus, the higher the outlier score of p. The higher p's outlier score, the higher the probability of p being an outlier.

B. Our Contribution: The Adaptive Threshold Updating Approach

The purpose of our approach (called VATU, variable-window-size adaptive threshold updating) is to handle the threshold setting procedure needed to detect outliers in a non-static data stream without requiring an inordinate amount of time and in the absence of labeled data beyond the initial set of background or normal data on which the anomaly detector is trained. We designed an informed adaptation method with the aid of a sliding window and a hypothesis test. The hypothesis test is used to detect when the threshold needs to be updated or not. The algorithm is shown in the Algorithm1 display. This algorithm is divided into two parts. It takes as input the training data, S, and values for each parameters. The training data contains only normal data.

The first part of the algorithm consists of obtaining the initial threshold (lines1-3). S is divided into a training data set and a validation data set. The training set is used to train a model (in this work, either an autoencoder or LOF), and the validation set is used to obtain the initial threshold. This threshold is set to the mean + 2 standard deviations of the outlier scores output by the model on the validation data. The getVlidationCost function aims to obtain the ouput (validationScores) of the classification model on each sample in the validation dataset.

The second part (lines 4-30) is designed to detect outliers in the data stream. When data point x_i is considered, we detect if it is an outlier according to the score obtained by the model. Three windows are used by the algorithm:t_w, z_w1 and z_w2. t_w is used for calculating the threshold. z_w1 and z_w2 are used to detect if the means of the scores in the two windows are significantly different. While both the t-test and the z-test can be used to assess whether a difference in means is significant, the t-test is used when the sample size is smaller than 30, whereas the z-test is used in other cases. Our algorithm uses the z-test as the number of scores considered before making a decision is always greater than 30. When the two windows z_w1 and z_w2 are full, we calculate the p-value derived from the z test on the two windows. If the p-value is smaller than 0.05, we update the threshold based on t_w. The calculateThreshold function calculates the threshold (mean+2std) based on the scores stored in z_w. In other words, we assume that when the mean of z_w1 is significantly different from that of z_w2, a concept drift has occurred. Hence, we re-calculate the threshold based on the new scores stored in t_w. After the update has occurred, we shrink t_w (line 19) so that it only covers the most recent scores. This allows us to forget what happened earlier in the stream and gets the window ready for the detection of potential future drifts in the data.

For safety, to guard against the case where the conditions for updating the threshold are never met, we automatically update the threshold when the size of t_w becomes divisible by 100 (lines 26-28). In such cases, however, we do not shrink the window since no concept drift has taken place and it would, therefore, be misguided to forget the past. Nonetheless, in order to decrease the consumption of storage, we do not let the window grow inordinately, we set an upper bound on the size of t_w. When the size of t_w reaches that upper bound, the oldest scores are dropped from t_w (lines 29-31) so that, we can guarantee that the size of t_w is not more than the upper bound.

Our algorithm is efficient since the calculation of the z-test and the updating of the threshold are not time consuming. Furthermore, our method does not rely on labeled data. In terms of functionality, we expect that, as we update the threshold using new scores, the number of false alarms will remain low even in the event of context drifts.

Algorithm 1 Adaptive Threshold for Outlier Detection on Data Streams

Input: initial training set S containing only data from a single class (the normal class), streaming data containing both normal data and outliers x1,x2,...., the window size for the z test is T=50, the window size for updating the threshold is M=100, The Upper Bound is 1000, The α value for the z test is α=0.05.

```
 1) f ← train_model (S (trainData))
 2) validationScores=getValidationCost(f,
    S(validationData))
 3) threshold ← 2 · std(validationScores) + mean (validationScores)
 4) $z\_w_1$ ← validation scores
 5) while $x_t$ is available
 6)     $score_t$ ← f($x_t$)
 7)     if $score_t$ > threshold
 8)         $\hat{\mathbf{y}}_t$ ← outlier
 9)     else
10)         $\hat{\mathbf{y}}_t$ ← normal
11)     $z\_w_2 \leftarrow z\_w_2 \bigcup score_t$
12)     $t\_w \leftarrow t\_w \bigcup score_t$
13)     end if
14)     if length($z\_w_1$) < T
15)         $z\_w_1 \leftarrow z\_w_1 \bigcup score_t$
16)     if length($z\_w_2$) ≥ T
17)         if z test($z\_w_1$, $z\_w_2$) < 0.05
18)             threshold ← calculateThreshold(t_w)
19)             t_w ← t_w[length(t_w) - T] to
    t_w[length(t_w)]
20)             $z\_w_1 \leftarrow z\_w_2$
21)             $z\_w_2$ ← clear window
22)         else
23)             $z\_w_2$.pop(0) //Remove first element
24)         end if
25)     end if
26)     if length(t_w) mod M = 0
27)         threshold ← calculateThreshold(t_w)
28)     end if
29)     if length(t_w) > Upper Bound
30)         t_w.pop(0)
31)         end if
32) end while
```

Fig. 1 illustrates the functioning of Algorithm 1 by describing the relationship between z_w1, z_w2 and t_w over time. z_w1, z_w2 and t_w are initialized as null. At the beginning, z_w1 stores the scores obtained on the validation data, z_w2 and t_w store the scores of the identified data. When the size of z_w2 reaches T, the z test is run, resulting in a p-value. If the p-value is smaller than 0.05, the scores of z_w2 are

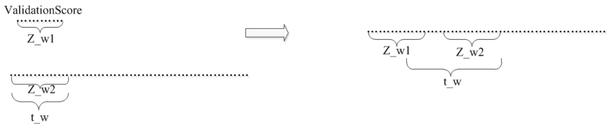

Fig. 1: relationship of z_w_1, z_w_2, and `t_w`

transferred to `z_w1`, `z_w2` is cleared, and `t_w` is shrunk. `z_w2` continuously stores the scores of the identified data up to a fixed size T. Once the z-test is run, the scores of `z_w2` are transferred to `z_w1`, and `z_w2` is cleared. `t_w` also grows continuously and continues growing as long as the z-test does not detect a significant difference and the window has not reached the upper bound. If the z-test does detect a significant difference in means, then `t_w` is shrunk and only stores the last T scores. Lastly, any time `t_w` reaches the upper bound, it drops the oldest score.

In algorithm 1, we also use a fixed window for the z test and threshold updating when the z test is not call if the scores are not significantly changed. Compared with the fixed-window based method (called FATU, fixed-window-size adaptive threshold updating) that updates the threshold when a fixed size of window is full, our method updates the threshold according to the z test. The variation of the size of the window may lead to significant change in the classification performance when using the fixed-window based method. Our method (VATU) can adaptively update the threshold according to the z test, so it will not be significantly influenced by the size of the window.

The inherent assumption of our method is that the distribution of the scores is Gaussian. It may lead to a 5% false alarm rate when there is no outlier. Therefore, our method is more suitable in situations where the scores follow a Gussian distribution and outliers exist in each window. In outlier detection, the other way to set the threshold is by chosing the top N score[16],which is similar to our method (we set top 5% as outliers). Please note that our training data does not contain abnormal(or outlier) data, it is difficult to set a good threshold. In addition, Our method mainly focus on threshold updating when concept drift happens, so as to adapt the change of the distribution of the score values. The algorithm shows that the threshold can be updated when the score values are siginificantly changed.

IV. Experimental Setup

This section is divided into two parts: in the first part, we describe the datasets we experimented on and in the second part, we describe our experimental settings. Subsequently, Section 5 discusses the results we obtained from our experiments.

A. Datasets

The data sets that we experimented on include two artificial datasets along with two real world data sets. The first artificial set was created using four Gaussian variables $\mu1$, $\sigma1$, $\mu2$, $\sigma2$ $\mu3$, $\sigma3$, $\mu4$, $\sigma4$. Over time $\mu1$,$\mu2$, $\mu3$, and $\mu4$ experienced an increase then decrease back down in two distinct ways. The first shift builds slow and gradual and the second is steeper and quicker shift. This was intended because larger windows perform more adequately in a time of stability and gradual change, and the a smaller window is optimal for the opposite situation in times of quick changes. The outlier class was also made of gaussian variables that experienced the same shifts as the normal class.[2]

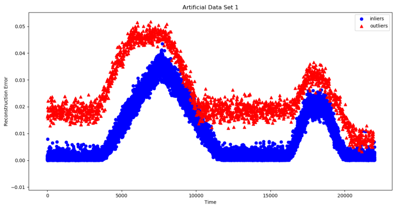

Fig. 2: Artificial 1

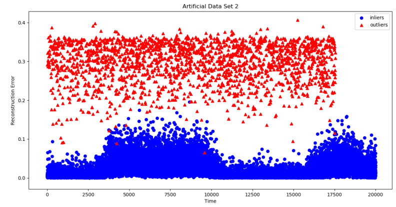

Fig. 3: Artificial 2

The second artificial set was created using a Beta distribution with parameters $\alpha1$ $\beta1$, $\alpha2$ $\beta2$,... The normal class experienced a shift up and down in α over time. The shifts have nearly the same changes, one gradual and one steep, similar to the first dataset. The outliers for this class were created in such a way that they would fall in or outside the tails of the inlier distributions. The differences between this set and the former is that the distribution is not gaussian and the outlier class in this example does not experience a shift.[2]

Each artificial data set consists of 90% of the normal class and 10% outliers. A visualization of the mentioned datasets are given in Figures 2 and 3 through the reconstruction errors of the autoencoder over time.

A last experiment was done on artificial data that represented the null model where there were no outliers or concept drift. The variables were created with the same parameters as the first artificial dataset with removal of a shift and outliers. The results for all methods show approximately a 95% accuracy and a 5% false positive rate.

The real datasets were taken from the UCI Machine Learning repository and the Stony Brook outlier detection database. We used the shuttle dataset from the first, and the waveform dataset from the second.

The shuttle data set contains statlog data from NASA that was generated over a time sequence. Approximately 80% of the data set belongs to class 1 and the following 20% belongs

[2] All datasets and programs are available at www.github.com/jc7553a/Adaptive_Threshold_DSAA

to the remaining 6 classes. We treated the majority class as normal and all others as outliers. To adjust this dataset we considered only 10% of the outliers and randomly put them into the time ordered normal class. The Stony Brook database offers a variety of different percentages of outliers in many data sets. For our study we chose the waveform data set that included 10% outliers.

B. *Experimental Settings*

We compared the performance of our algorithm (called VATU, variable-window-size adaptive threshold updating), which uses a variable sized sliding window, against a baseline that does not adjust its threshold, as well as an algorithm that does adjust its threshold but uses a fixed size window of size n to calculate the new threshold (called FATU, fixed-window-size adaptive threshold updating). We experimented with n=100 and n=500. For all experiments, the threshold was calculated as $\mu + 2\sigma$. As discussed earlier, the window sizes for the z-test in all experiments involving our algorithm were set to 50. The upper bound on the size of the window our algorithm used for threshold determination was set to 1000. The amount of training data used in both artificial data sets were the first 500 instances. The remaining 20000 data points were used for testing. Training for the waveform data set dealt with the first 1000 examples. Lastly on shuttle we used the first 10% of the dataset for training.

As previously discussed, we used both an Autoencoder and LOF as our one-class learning anomaly detection systems. The autoencoders all used a hidden layer whose size was half the size of the given input for all experiments. The learning rate for all experiments was held at .05, and the activation function was the sigmoidal function. The number of epochs for each training set was 20 for all experiments.

All of LOFs settings corresponded to the default settings for Wekas LOF Class implementation in Java. This included a lower bound of 10 neighbors and an upper bound of 40 neighbors. The nearest neighbor search algorithm was set to the linear nearest neighbor search.

The performance metrics chosen for all experiments are the F-measure and the false alarm rate. We chose the F-measure in order to show that we are able to catch outliers while keeping a low false alarm rate. We chose to report on the false alarm rate because the challenge of streaming data with concept drifts is the increase in false alarms. This is because data that has drifted looks like anomalies to a static anomaly detection system. Therefore, the main purpose of our research is to show how the adaptive threshold our method calculates is able to account for the detection of outliers whilst keeping a low false alarm rate.

V. Experimental Results

We begin by presenting and discussing the results we obtained on our experiments using the anomaly detectors, parameters and domains stated in Sections 3 and 4. We then experiment with different parameter values and discuss these results. Altogether, as expected, our results show that our adaptive threshold setting algorithm allows us to lower the false alarm rate of the baseline method which consists of setting a threshold only once at deployment time. More significant, however, is the fact that our method displays the kind of flexibility that makes it fullproof as compared to fixed-window-size updating which works well on some domains for certain window sizes but badly on others and vice-versa, making such approaches difficult to deploy, in practice. In the following figures, Adaptive window refers to our algorithm (i.e.,VATU), Fixed Window refers to the fixed window size based algorithm(i.e. FATU), and baseline refers to the the static model without threshold updating.

Algorithm	FMeasure	False Alarm
VATU(AE)	.935	.0026
FATU (500, AE)	.843	.005
FATU (100, AE)	.934	.0006
Baseline (AE)	.275	.585
VATU (LOF)	.887	.0064
FATU (500, LOF)	.911	.0034
FATU (100, LOF)	.881	.0062
Baseline (LOF)	.396	.339

TABLE I: Artificial Data Set 1

Algorithm	FMeasure	False Alarm
VATU (AE)	.887	.0064
FATU (500, AE)	.911	.0034
FATU (100, AE)	.881	.0062
Baseline(AE)	.396	.339
VATU (LOF)	.922	.005
FATU (500, LOF)	.923	.006
FATU (100, LOF)	.908	.0062
Baseline(LOF)	.303	.511

TABLE II: Artificial Data Set 2

A. *Comparative Results*

Tables I and II show the F-measure and false alarm results obtained by the autoencoder and the LOF on artificial data sets 1 and 2. Higher F-measures and lower false alarm rates are the desired outcomes.

Looking at the F-measure results, first, we can see that, as expected, the results obtained by the baseline algorithm which never updates the threshold are a lot worse than by the other three algorithms. Furthermore, we can see that VATU, our algorithm, performs either better than all the other methods or very closely to the best one on both domains and when using both anomaly detectors. What is interesting, however, is that in the case of the autoencoder, where the FATU is slightly better than VATU, the best window size, in the fixed-window size case is different for each artificial domain: it is 100 in the case of the first domain and 500 in the case of the second. Yet in both cases, VATUs performance approaches the performance of the winning window size.

Since choosing a fixed window size is an extremely difficult task prior to deployment, VATU can relieve the burden of this problem. Furthermore, since the window grows and shrinks over time it is able to adapt and act similarly to any size fixed

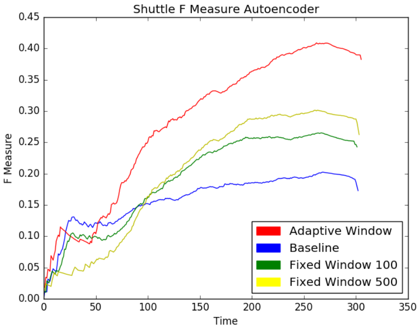

Fig. 4: F-measure of Autoencoder on Shuttle Data

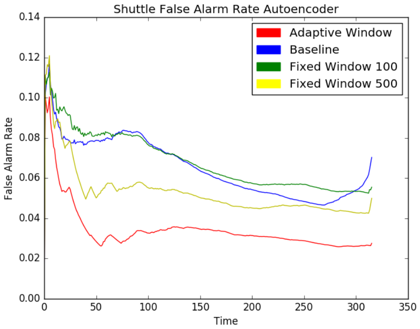

Fig. 5: False alarm of Autoencoder on Shuttle Data

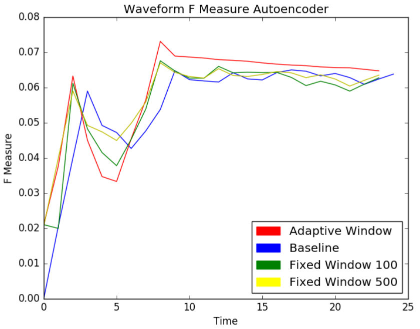

Fig. 6: F-measure of Autoencoder on Waveform Data

window that is suitable for the given problem and, in fact, can adapt to cases where the optimal initial window size differs from the most appropriate one at a later time.

Figures 4 to 7 show the F-measure and false alarm obtained by the autoencoder on the Shuttle and Waveform data sets. Figures 8 to 11 show the F-measure and false alarm obtained by LOF on the same data sets. The results show that on both datasets, the autoencoder and LOF are, both, capable of outperforming the baseline and FATU strategies when using the VATU method.[2]. This shows that our approach is valuable

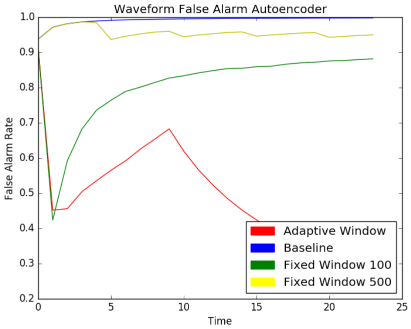

Fig. 7: False alarm of Autoencoder on Waveform Data

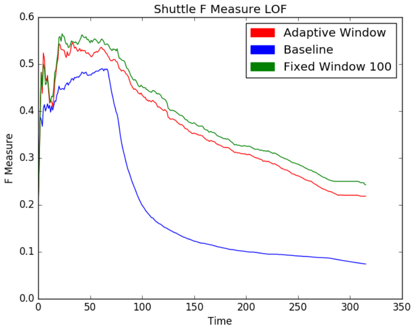

Fig. 8: F-measure of LOF on Shuttle Data

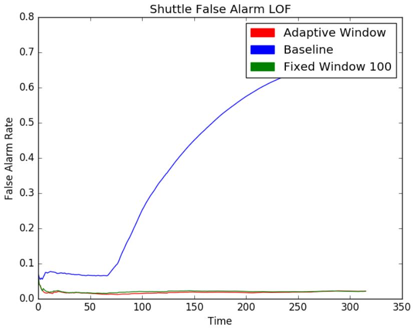

Fig. 9: False alarm of LOF on Shuttle Data

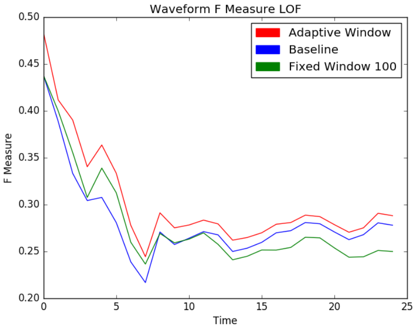

Fig. 10: F-measure of LOF on Waveform Data

[2] Although it looks like the baseline performs better than all the other algorithms in Figure 11 (LOF, Waveform, False Alarm Rate), a look at the scale of that graph shows that the differences between each method is, actually, insignificant.

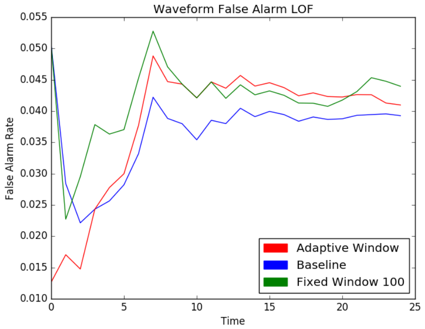

Fig. 11: False alarm of LOF on Waveform Data

in real-life application settings and not, simply, in artificially created domains.

We now focus on the results obtained by the autoencoder using VATU. We see that that approach outperforms the baseline for all the given data sets except on one of the artificial sets where the fixed window size beats the VATU by a small margin. This may be due to the presence of a quick distribution changes in the data (when there are many shifts in the data, the smaller the window size, the quicker the threshold will adapt to new shifts). On both the shuttle and waveform datasets, the autoencoder outperforms both FATU and the baseline on both metrics. It leads us to believe that for outlier detection using a combination of autoencoder and the VATU method is a very good choice. However, to be fair, it should be noted that the results obtained by the autoencoder on the real-world domains overall are quite low (in terms of F-measures, not much higher than .4 for the Shuttle dataset and not much higher than .07 for the waveform dataset). While it is important to keep in mind that these results were obtained without any labeled data, it is important to note that under the same restrictions, LOF performs better on the real-world data sets as discussed below.

We now focus on the results obtained by LOF using VATU. We see that this approach shows similar patterns as with the autoencoder. In particular, LOF outperforms the baseline on all of the artificial domains. It is also shown to have a low false alarm rate on the second artificial data set, although it has a high one on the first. This could be due to the nature of LOFs underlying algorithm where it chooses outliers based on density measurements. Considering that the outlier class is made of gaussian random variables, they will have a higher density in more compact areas than from a uniform distribution. When it comes to the shuttle and waveform datasets, LOF outperforms the autoencoder on both as it gets over the 0.5 F-measure range for the Shuttle data and in the .25 to .45 range for the Waveform data. For LOF, VATU outperforms the fixed threshold baseline and FATU on the waveform data set, and it falls very close to the best algorithm, the fixed window size of 100, on the shuttle dataset.

B. Sensitivity Analysis

1) Parameter Value M: The parameter value, M, in our algorithm is a user defined parameter that updates the threshold every M instances in case no significant difference were detected by the z-test. In the experiments just reported, M was set to 100. Since, as we saw previously, setting the window size in FATU was important, the purpose of this section is to show that the VATU does not exhibit the same dependence on parameter M, which could be considered related to the fixed window size. To show that our method is robust and not too sensitive to the value chosen for M, we compared the results obtained by VATU with M=100 and M=500 to the results obtained by the FATU with window sizes of 100 and 500. The results are shown in Figures 12 to 15.

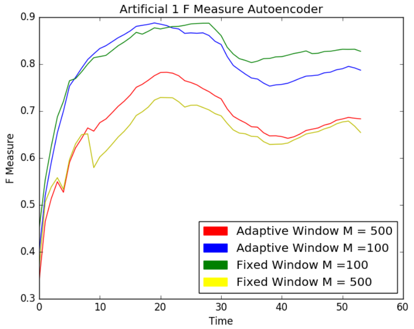

Fig. 12: F-measure with different M on Artificial Set 1

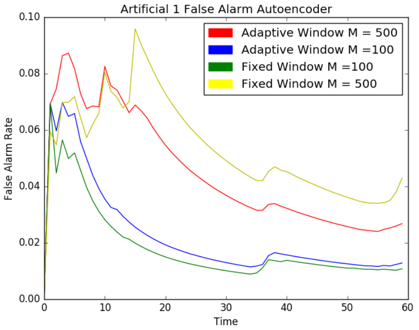

Fig. 13: False alarm with different M on Artificial Set 1

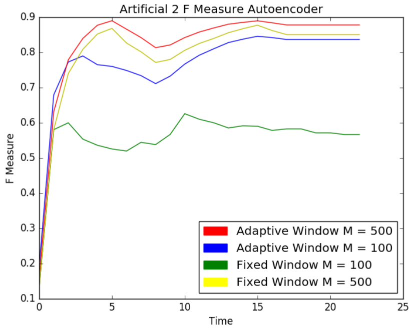

Fig. 14: F-measure with different M on Artificial Set 2

The results show that our method is sensitive to the value of M but not nearly as sensitive as FATU is to the size of the window. Indeed, while there is a drop in performance when

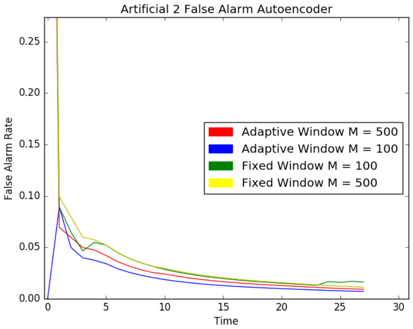

Fig. 15: False alarm with different M on Artificial Set 2

the inappropriate M-value is used, that drop is not as dramatic as in the case of FATU.

2) Parameter Value T: The parameter value T represents the size of the windows used in the z-test by our algorithm. In all the experiments the value chosen for T was 50. We used 50 since it is greater than 30 which the central limit theorem considers the minimal number of instances needed to consider their distribution to be approximately normally distributed and the values to be independent, the two required assumptions of the z-test.

3) Parameter Value Upper Bound: The upper bound chosen in all experiments was set to 1000. The purpose of this parameter is to set an upper bound on the size of VATUs threshold window. If the z-test was never to find a significant difference, it would mean that no concept drift took place. If this were the case, the window from which thresholds are calculated would grow intractably. To keep computation costs low and bound the space required by our algorithm, we used an upper bound at which the threshold gets automatically re-calculated and the window size reset to its initial size.

VI. Conclusions

Outlier detection algorithms have attracted a lot of attention in recent years. In the area of outlier detection applied to data steams, most research focuses on updating the distances or retraining the one class classification model. Such approaches are time consuming, and may worsen the outcome of the algorithm when no label information is available in the data stream. A better alternative could be not to re-train the detector, but instead to adjust its threshold. For algorithms with outlier score outputs, a static threshold may lead to high false alarm. To handle this problem, this paper describes an adaptive threshold setting method for data streams. Our approach updates the threshold when significant changes in the mean of the outlier scores occurs as reported by the z test. Our approach is unsupervised and efficient and our experimental results show that it is better than two other approaches in terms of F-measure and false alarm. To further improve the performance of our approach, we propose to use ensembles of anomaly detectors and design a threshold-setting approach in such a context. We are also planning to integrate concept drift detection methods other than the z-test to our algorithm.

Acknowledgment

We thank the anonymous reviewers for their constructive comments. This work is supported by the National Natural Science Foundation of China under Grant No. 61501128, financial support from China Scholarship Council and a grant from American University.

References

[1] Knorr, E. M., Ng, R. T., & Tucakov, V., Distance-based outliers: algorithms and applications., The VLDB Journal - The International Journal on Very Large Data Bases. 8(3-4), 237-253, 2000.

[2] Zhang, K. Hutter, M., & Jin, H.,A new local distance-based outlier detection approach for scattered real-world data, In Pacific-Asia Conference on Knowledge Discovery and Data Miningm pp.813–822, Springer, Berlin, Heidelberg.

[3] Breunig, M. M., Kriegel, H. P., Ng, R. T., & Sander, J. (2000, May). LOF: identifying density-based local outliers. In ACM sigmod record (Vol. 29, No. 2, pp. 93-104). ACM.

[4] Loureiro, A., Torgo, L., & Soares, C. (2004). Outlier detection using clustering methods: a data cleaning application. In Proceedings of KDNet Symposium on Knowledge-based Systems for the Public Sector. Bonn, Germany.

[5] Ma, Y., Zhang, P., Cao, Y., & Guo, L. (2013, October). Parallel auto-encoder for efficient outlier detection. In Big Data, 2013 IEEE International Conference on (pp. 15-17). IEEE.

[6] Tran, L., Fan, L., & Shahabi, C. (2016). Distance-based outlier detection in data streams. Proceedings of the VLDB Endowment, 9(12), 1089-1100.

[7] Toshniwal, D. (2012). A framework for outlier detection in evolving data streams by weighting attributes in clustering. Procedia Technology, 6, 214-222.

[8] Pokrajac, D., Lazarevic, A., & Latecki, L. J. (2007, March). Incremental local outlier detection for data streams. In Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on (pp. 504-515). IEEE.

[9] Dong, Y., & Japkowicz, N. (2018). Threaded ensembles of autoencoders for stream learning. Computational Intelligence, 34(1), 261-281.

[10] Swersky, L., Marques, H. O., Sander, J., Campello, R. J., & Zimek, A. (2016, October). On the evaluation of outlier detection and one-class classification methods. In Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on (pp. 1-10). IEEE.

[11] Dani, M. C., Jollois, F. X., Nadif, M., & Freixo, C. (2015, November). Adaptive threshold for anomaly detection using time series segmentation. In International Conference on Neural Information Processing (pp. 82-89). Springer, Cham.

[12] Zhang, Y., Meratnia, N., & Havinga, P. (2010). Outlier detection techniques for wireless sensor networks: A survey. IEEE Communications Surveys & Tutorials, 12(2), 159-170.

[13] Rosner, B. (1983). Percentage points for a generalized ESD many-outlier procedure. Technometrics, 25(2), 165-172.

[14] Manevitz, L. M., & Yousef, M. (2001). One-class SVMs for document classification. Journal of machine Learning research, 2(Dec), 139-154.

[15] Japkowicz, N., Myers, C., & Gluck, M. (1995, August). A novelty detection approach to classification. In IJCAI (Vol. 1, pp. 518-523).

[16] Cao, L., Yang, D., Wang, Q., Yu, Y., Wang, J., & Rundensteiner, E. A. (2014). Scalable distance-based outlier detection over high-volume data streams. IEEE 30th International Conference on Data Engineering,pp. 76-87.