DBOD-DS: Distance Based Outlier Detection for Data Streams

Md. Shiblee Sadik, Le Gruenwald

School of Computer Science, University of Oklahoma
110 W. Boyd St, Norman, OK 73019, USA
{shiblee.sadik, ggruenwald}@ou.edu

Abstract. Data stream is a newly emerging data model for applications like environment monitoring, Web click stream, network traffic monitoring, etc. It consists of an infinite sequence of data points accompanied with timestamp coming from external data source. Typically data sources are located onsite and very vulnerable to external attacks and natural calamities, thus outliers are very common in the datasets. Existing techniques for outlier detection are inadequate for data streams because of its metamorphic data distribution and uncertainty. In this paper we propose an outlier detection technique, called Distance-Based Outline Detection for Data Streams (DBOD-DS) based on a novel continuously adaptive probability density function that addresses all the new issues of data streams. Extensive experiments on a real dataset for meteorology applications show the supremacy of DBOD-DS over existing techniques in terms of accuracy.

Keywords: Data stream, outlier detection, probability density function.

1 Introduction

Applications like environment monitoring, Web click stream, and network traffic monitoring use a new data model to represent their never ending series of data called data streams. Data stream has received a great deal of attention in the research community in recent years due to its novel characteristics. On the other hand every real life dataset has outliers in it [6]; therefore outlier detection is a very important part of data acquisition. In most of the cases the work done on outlier detection for data streams [1], [3], [8] is adopted from outlier detection techniques for regular data with ad-hoc modifications and do not address all the novel characteristics of data streams. In this paper we propose a novel outlier detection technique to fill the gap. Before going further we briefly discuss the novel characteristics of data streams and data stream processing requirements.

Applications for data streams are significantly different from those for regular data in many facets. In data stream applications, data have the essence of time, are mostly append only and, in many cases, are transient [2], [5]; therefore offline store and process approaches are not very suitable for online data stream; consequently data processing has to be online and incremental [25]. Data are continuously coming in a streaming environment in a very fast rate with changing data distribution [17], and thus any fixed data distribution is not adequate enough to capture the knowledge. On top of this, in many cases uncertainty in data streams makes processing more complicated. The novel characteristics of data streams bring the outlier detection problem out on the open again. The next paragraph introduces the problem of outlier detection in a real life dataset.

An outlier refers to a data point which does not conform well to the pattern of the other data points or normal behaviors or conform well to the outlying behavior [4], [6]. Pragmatically, normal behaviors are easy to identify and every possible outlying behavior are difficult to compile; nonetheless the outlying behaviors are changing over time. Almost all real datasets have outliers [6]. The major reasons behind the outliers are malicious activity or intrusion, instrumental error or setup error, change in environment, human error, etc. Evidently, outlier detection is not a new topic at all. It has been in the literature since the eighteenth century [4]. Even though the problem has been in the literature for so many years it is still very popular; this is because nobody knows the real outliers and the detection of outliers is very subjective to the application. The outlier detection with perfect confidence in regular data is still not an easy problem. This is because of the inherent vagueness in the definition of outlier, like how to define regular behavior, to what extend an outlier needs to be not conforming to the regular behavior, etc. The problem of outlier detection becomes more complicated when considering new characteristics of data streams, such as unbounded data, varying data distribution, data uncertainty, and temporal dimension. None of the existing outlier detection techniques addresses all of these characteristics. In this paper, we present a novel outlier detection technique for data streams based on the concept of probability density function, called Distance-Based Outlier Detection for Data Streams (DBOD-DS), that addresses all the characteristics of data streams. We then present the results of the experiments that we have conducted on a real dataset obtained from a meteorological data stream application [7] to compare the accuracy and execution time of DBOD-DS with the two outlier detection techniques existing in the literature: ART [8] and ODTS [3].

The rest of the paper is organized as follows: Section 2 discusses the work related to outlier detection in data stream; Section 3 describes our approach and its implementation; Section 4 presents the experimental results we have obtained, and finally Section 5 provides our conclusions and future research.

2 Related Work

Most of the outlier detection techniques for data streams use a sliding window to capture the recent data values and detect the outliers inside the window [1], [3], [26] with multi-pass algorithms. Data streams change over time and an outlier for a particular window may appear as an inlier in another window; hence the notion of outlier in a data stream window is not very concrete. Nevertheless, an inlier can be shown as an outlier by changing the window size [3]; thus the outlier detection techniques that use a sliding window work well if the window size is chosen carefully. However, different techniques interpret window size differently; in most situations, it is difficult for the domain expert to choose the window size correctly without knowing the interpretation of a particular technique.

Auto-regression based techniques for outlier detection are very popular for time series outlier detection [4]. Some outlier detection techniques for data streams adopt auto-regression [8], [22]. Most of the auto-regression based techniques work similarly in which a data point is compared with an estimated model and a metric is computed based on the comparison. If the metric is beyond a certain limit (called cut-off limit), the data point is identified as an outlier. The advantages of auto-regression based models are that they are computationally inexpensive and they provide an estimated value for the outlier. However, the success of this method depends on the quality of the auto-regression model and the efficacy of the cut-off limit. Different data streams show different natures in their changing patterns; therefore it is very difficult to select an appropriate auto-regression model for data streams [8]. The selection of a magic cut-off point not only depends upon the data but also the auto-regression model chosen.

Outlier detection techniques for multiple data streams have been proposed in the literature [16], [10], [11], [26]. The underlying assumptions are the availability of multiple homogeneous data streams and their synchronous behavior. These may not be the case as multiple homogeneous data streams may not be available or one data stream may behave very differently from the others. In the later case comparing two heterogeneous data streams does not help to point out the outliers.

Statistical [4] and machine learning [9] based techniques assume a fixed distribution for the data and if the probability of a data point is very low it is identified as an outlier by statistical and machine learning based techniques. Data streams are highly dynamic in nature and their distribution changes over time. No fixed data distribution is good enough for the entire data stream; hence summarizing a dynamic data stream with a static data distribution produces questionable results.

Data clustering algorithms produce outliers as a bi-product [21], [24]; but as outlier detection is not the focus of clustering algorithms, they are not optimized for outlier detection. Keogh *et al* argued that most of the clustering algorithms for time series/data stream produce meaningless results [18]; hence their efficacy and correctness are still in question.

However none of the existing outlier detection technique considers the uncertainty, concept drift and the transient property of the data stream. Moreover, not all the outlier detection algorithms are truly incremental rather they store a subset of the data points and use multi-pass algorithms to detect the outliers in the subset. While designing a technique of outlier detection for data streams, one needs to consider the uncertainty, the drift of concepts, the transient property, the temporal characteristic of the data points, etc. On top of this, every computation has to be online and incremental. To fill the gap, we have designed our technique addressing the fact that data points in a data stream are very uncertain. We also address temporal characteristics of the data points. Moreover we do not assume any type of fixed data distribution to address the fact that the concept drift occurs in data stream. Next section (3) portrays the details of our algorithm with the implementation issues.

3 Proposed technique: Distance-Based Outlier Detection for Data Streams (DBOD-DS)

In this section, we first provide an overall description of our proposed technique, DBOD-DS. We then discuss our novel probability density function, which is the basis of our technique, and algorithms to implement it.

3.1 Overall Approach

Our approach is motivated by distance-based outlier [26], [19] and based on a probability density function $f(x)$ which resembles data distribution where x is a random variable. As each data point d with the value v comes in we compute the probability of occurrence of the values $p(v,r)$ within user defined radius r from the data value v ($p(v,r)$) by integrating the probability density function $f(x)$ from $v-r$ to $v+r$, $p(v,r)=\int_{v-r}^{v+r} f(x)dx$. The probability of occurrence resembles the neighbor density around the data value [19]; if the neighbor density is very low the data point is more likely to be an outlier. According to our approach, if the probability of occurrence $p(v,r)$ is less than the user defined minimum probability of occurrence (q) i.e., $p(v,r) < q$ the data point d is identified as an outlier.

As we receive each data point d, we update the probability density function $f(x)$ by increasing the probability of occurrence of data value v. To address the data uncertainty characteristic of data stream, when we receive the data point d we not only increase the probability of the data value (v) by p_v but also increase the probability of other values by a fraction of $(1-p_v)$ where p_v is the probability of occurring the data value v while there is a data uncertainty.

To address the temporal characteristic of the data streams, when we compute the probability density function $f(x)$ the data points $(d_1, d_2, \dots, d_n)$ are weighted based on their freshness. The most recent one receives the highest weight and the oldest one receives the lowest weight. If the respective values are $(v_1, v_2, \dots, v_n)$ where the v_n is the most recent one and v_1 is the oldest one, we weight them by $(\lambda^{n-1}, \lambda^{n-2}, \dots, 1)$, respectively; therefore for the value (v_i) we update the probability density function $f(x)$ by increasing the probability of occurrence of v_i by $p_{v_i}\lambda^{i-1}$ and the probability of others values by a fraction of $(1-p_{v_i})\ \lambda^{i-1}$.

To address the varying data distribution characteristic of data streams, our probability density function $f(x)$ does not assume any particular fixed data distribution; rather we adjust our probability density function on-the-fly; therefore our probability density function ($f(x)$) never becomes obsolete due to a change in data distribution (concept drift [17]), rather our probability density function ($f(x)$) always provide the most recent data distribution.

Now at any particular time if we integrate our probability density function from $v-r$ to $v+r$ we obtain the probability of occurrence $p(v,r)$ of a data value v within $v-r$ to $v+r$. If $p(v,r)$ is large, then the data value v has a very high probability of occurrence or neighbor density in recent time. Therefore our approach requires two user defined parameters, radius r and minimum probability of occurrence q. However if the probability function density function $f(x)$ is continuous, the same result can be produced by a different set of (r, q), thus by fixing the value of q and changing the value of r we can obtain the optimal result, which reduces the curse of having two parameters to one. We fix the value of q and change the value of r to receive the optimal performance. The next section (3.2) presents the detail of our proposed probability density function.

3.2 Proposed Probability Density Function

Our proposed probability density function is based on a kernel probability density estimator. Several techniques exist in literature to estimate probability density function like histogram [15], wavelet [14], kernel estimation [26], etc. Among those techniques we choose kernel probability density estimator (in short kernel estimator) for our approach. We will justify our choice in the next paragraph.

The kernel estimator estimates the probability density function based on the data values. For each data value v the kernel estimator increases the probability of occurrence of v by p_v and increases the probability of occurrence of other values by a fraction of $1 - p_v$ which fits our requirements excellently. Due to data uncertainty when we receive a data point d with value v, we cannot assert the data value with full confidence; therefore we cannot increase the probability of occurrence of v by 1. Since the value v is uncertain, it might be induced by other data values other than v. Thus to address the uncertainty of data streams, we do not increase the probability of occurrence of v by 1. Kernel estimator increases the probability of occurrence of v by p_v and distributes the rest of the probability of occurrence $(1 - p_v)$ into the other data values which are close to the value v. Formally, if $(x_1, x_2, \dots, x_n)$ are n sample data points, their respective values are $(v_1, v_2, \dots, v_n)$ and the probability density function $f(x)$ is defined by equation (1) where $k(x)$ is called the *kernel function*. v_i can be a scalar or vector.

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} k(v_i - x) \tag{1}$$

The kernel function is responsible for distributing the probability of occurrence induced by the data value v_i. Various researchers have proposed various kernel functions (e.g., Uniform kernel function, Triangle kernel function. Epanechnikov kernel function, Normal kernel function etc. [23]). Different kernel function distributes the probability of occurrence differently. Interestingly, the choice of a kernel function does not affect the probability density function very much [23], [26]. Typically the kernel function distributes the probability of occurrence into the neighbor data values which reside within a range called *bandwidth* (h) (Normal kernel function distributes the probability of occurrence from $-\infty$ to $+\infty$ [23]). A kernel function along with the bandwith (h) (is denoted by $k_h(x)$ where $k(x) = hk_h(x)$. Although the choice of the kernel function is not very significant, the choice of the bandwidth is very important for probability density function estimation. A detailed discussion about the choice of kernel function and bandwidth selection can be found in [23]. In our approach we choose a data-based approach for bandwidth selection. Scott's rule provides a data-based bandwidth selection where $h = \sqrt{5}\sigma n^{-1/5}$ where σ

is the standard deviation and n is the number of data points used for density estimation [26].

In a kernel estimator the probability of occurrence is distributed into the equal number of neighbor values for each data point, but in a variable kernel estimator the probability of occurrence is distributed into different number of neighbor values for each data point. Hence at any specific point of time, if data values are close to each other (in terms of value) the bandwidth becomes small and if the data points are far (in terms of value) from each other the bandwidth becomes large. Let $(x_1, x_2, \ldots, x_n)$ be our data points with values $(v_1, v_2, \ldots, v_n)$ at times $(T-n, T-n+1, \ldots, T)$, and our corresponding bandwidths be $(h_1, h_2 \ldots, h_n)$. The probability density function $(f(x))$ at time T becomes equation (2) where $f_T(x)$ is the probability distribution function at time T. In our approach we use variable kernel estimator.

$$f_T(x) = \frac{1}{n}\sum_{i=1}^{n} k_{h_i}(v_i - x) \tag{2}$$

The use of variable kernel estimator is twofold: the variable kernel estimator offers variable bandwidth for each data points, therefore the bandwidth can be computed on-the-fly using Scott's rule for each data point and the variable kernel selects the bandwidth based on recent data values only.

We modify the variable kernel estimator to address the temporal characteristic of data streams. Recent data points are more interesting than old data points; therefore, when we estimate the probability density function we need to consider the freshness of data points. Heuristically, the recent data items should have more *weight* than the old data points [22], [20], [27]. Here weight is defined as how a data point contributes to the probability density function; thus, in our probability density function, instead of giving all data points the same weight we weight them according to their freshness. The most recent data point receives the highest weight while the oldest one receives the lowest weight. Exponential forgetting is a weight assigning scheme which gives more weight to the recent data points and less weight to the old data points and the weight is decreasing exponentially from present to past [28]. According to exponential forgetting the relative weight among two consecutive data points is constant, called forgetting factor (λ) where $0 < \lambda \leq 1$. Among the two consecutive data points, the recent data point receives weight 1 and the old one receives weight λ. In case of a series of data points, at any particular time the most recent data point receives the weight 1 and all other data points receive the weights according to their relative positions to the most recent data point. If $(x_1, x_2, \ldots, x_n)$ are the data points with data values $(v_1, v_2, \ldots, v_n)$, at time $(T-n, T-n+1, \ldots, T)$ respectively, the corresponding weights are $(\lambda^{n-1}, \lambda^{n-2}, \ldots, 1)$. We weight the kernel function with an exponential forgetting factor. Adding the exponential forgetting factor λ to the equation (2), the probability density function becomes equation (3) where $\sum_{i=1}^{n} \lambda^{n-i}$ is the total weight.

$$f_T(x) = \frac{\sum_{i=1}^{n} \lambda^{n-i} k_{h_i}(v_i - x)}{\sum_{i=1}^{n} \lambda^{n-i}} \tag{3}$$

One advantage of using exponential forgetting factor is that it can be computed incrementally, which eases one-the-fly implementation for data streams [28]. The λ is the parameter which decides how many data points contribute to the probability estimation; the value 0 implies no history, only the previous data point, while value 1 implies all previous data points. Brailsford *et al* [28] proposed a λ selection scheme based on a bootstrap method; we adopt this approach for λ selection. The details about and λ selection are omitted due to page limitation, the detail can be found in [28]. The next section (3.2) discusses the online implementation of our proposed probability density function.

3.3 Implementation of Proposed Probability Density Function

The kernel estimator requires a large amount of computation. Binned implementation is a popular, fast implementation for the kernel estimator [13]. In this approach the entire range of data points is divided into some equally spaced bins and data are distributed into bins. Each bin has a representing value and all the data point in a bin are represented by the representing value. The key idea is that lots of values are practically close to each other and binned implementation reduces the number of evaluations; but this popular binned implementation still requires multiple passes and cannot be computed incrementally.

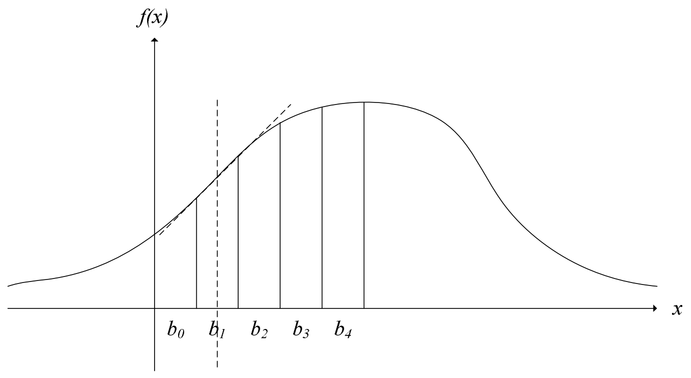

Fig 1. Binned implementation of kernel estimator

In our approach we also divide the entire range of data values into equally spaced bins. A representing value is selected for each bin ($b_0, b_1, b_2, \dots$ in the Figure 1). Instead of binning the data points, for each bin, we store the value of probability density function of the representing value b_i, $f(b_i)$ and the derivative of the probability density function $f'(b_i)$. $f(b_i)$ and $f'(b_i)$ are stored for each representing value b_i. $f(b_i)$ and $f'(b_i)$ are the sum of the value of the kernel function and the sum of the derivative of the kernel function at representing value b_i, respectively. The kernel function and the derivative of the kernel function for each representing value are computed on-the-fly and added to the previous sum; hence this is an online incremental implementation.

Fig 1 shows the binned implementation of our proposed probability density function. By carefully selecting the bin width we can assume each bin is a trapezoid as shown in Figure 1 and we can approximate the probability of any value within a bin. The top of the trapezoid is a straight line (shown in Figure 1 as the dotted line touching the probability density function) and we store the passing point as well as the derivative; hence using the straight line equation of the line we can estimate the probability of occurrence of any data value within a bin. The bin width should be such that the average error is minimum. Fan and Marron [13] stated that four hundred bins is often optimal, fewer than four hundred bins often deteriorates the quality of the results and more than four hundred bins offer very little improvement. In our approach we use the optimal four hundred bins. Due to page limitation we omit the detail discussion about bin width selection.

The data structure for binned implementation of probability density function is composed of grid cells. As each time a data point comes in, we update the necessary grid cells on-the-fly. Each cell corresponds to a bin. Each cell contains the value of probability density function at b_i, $f(b_i)$, derivative of the probability density function $f'(b_i)$ at b_i and the timestamp (t) when the cell is last updated. The next section (3.3.1) describes the algorithms for updating the probability density function using our data structure and computing the probability of occurrence of a data value.

3.3.1 Algorithms

Figure 2 shows the online incremental update and probability of occurrence lookup algorithms for our proposed probability density function and outlier detection technique. The update algorithm updates the data structure as each data point comes in and the probability computation algorithm computes the probability of occurrence of a given value (x). The update algorithm takes a data point and its timestamp as input. It starts with the updating of the weighted summation (lines 2 & 3), where s_1 is the weighted summation of the data values and s_2 is the weighted summation of the square of the data values. The ω in line 4 is the total weight of the data. s_1 and s_2 are required to calculate the current standard deviation (σ) and hence the bandwidth (h). In line 9 we calculate the number of cells we need to update. Some kernel function updates the values in the range from $-\infty$ to ∞ (e.g., Normal kernel function [23]); in that case we restrict it to $minValue$ and $maxValue$, which represent the minimum and maximum allowable values for a data point, respectively. Now for each bin we update the sum of the kernel function and the latest timestamp when the bin is updated. If the kernel function is continuous at the representing point (b_i) then we store the derivative of the kernel function at b_i else we store the gradient from the starting point (α_i) to the end point (β_i) of the bin. The probability lookup algorithm is fairly simple; it finds the appropriate bin which contains the sum of the kernel function values. Finally the probability is achieved by dividing the sum of the kernel function values by the total weights.

```
procedure update(dataItem d, timestamp t)
    $s_1 \leftarrow \lambda s_1 + d$; // $m_1$ is the sum of data value and $\lambda$ is our forgetting factor
    $s_2 \leftarrow \lambda s_2 + d^2$; // $m_2$ is the sum of the square of the data value
    $\omega \leftarrow \lambda\omega + 1$; // $\omega$ is the total data weight
    $\mu_1 \leftarrow s_1/\omega$; // $\mu_1$ the first moment
    $\mu_2 \leftarrow s_2/\omega$; // $\mu_2$ the second moment
    $\sigma \leftarrow \sqrt{\mu_2 - \mu_1^2}$; // $\sigma$ is the standard deviation
    $h \leftarrow \sqrt{5}\sigma\omega^{-1/5}$; // h is the bandwidth
    $c \leftarrow h/binWidth$; // c is the cell count
    $b \leftarrow indexLookup(d)$; // b is the middle cell
    for $i = b - c$ to $b + c$, // i is the index of the cell, where $i \geq 0$ and $i \leq maximum\ index$.
      // $b_i$ is the representing value of the bin/cell($c_i$) and $\alpha_i$ and $\beta_i$ are the starting value and the end value of the bin.
      // distance between two consecutive time stamp is 1.
      $c_i[f(b_i)] \leftarrow \lambda^{(t - c_i[timestamp])} c_i[f(b_i)] + k_h(d - b_i)$;
      if ($k_h(d - x_i)$ is not discontinuous at $x_i$
        $c_i[f'(b_i)] \leftarrow \lambda^{(t - c_i[timestamp])} c_i[f'(b_i)] - k'_h(d - b_i)$;
      else
        $c_i[f'(b_i)] \leftarrow \lambda^{(t - c_i[timestamp])} c_i[p'(b_i)] - (k_h(d - \beta_i) - k_h(d - \alpha_i))/binWidth$;
      $c_i[timestamp] \leftarrow t$;
    end for
end procedure
procedure indexLookup(dataItem d)
    return $\lfloor (d - minValue)/binWidth \rfloor$;
end procedure
procedure probability(x)
    $i \leftarrow indexLookup(x)$;
    $f(x) \leftarrow (c_i[f(b_i)] + c_i[f'(b)](x - b_i))/\omega$;
return $p(x)$;
```

Fig 2. Update and probability of occurrence lookup algorithm

4 Performance Analysis

We conducted experiments using a real dataset collected from California Irrigation Management to compare the performance of our algorithm in terms of detection accuracy and execution time with that of the two existing algorithms: We compare our algorithms (DBOD-DS) with two other algorithms ART [8] and ODTS [3] from the literature. ART is an auto-regression based outlier detection technique for wireless sensor network which estimates the value using an auto-regression model and compare the estimated value with the data value received; if the distance is greater than a user defined threshold the data point is identified as outlier., The ODTS is an outlier detection technique for time series. It uses a sliding window to store the recent subset of the data and compare each data point with median value, if the distance is greater than user defined threshold the point is identified as outlier. Since ODTS uses a sliding window, we run the experiments with the window sizes $10, 15, 20, \ldots, 100$ and report the average performances. In this section, we first describe the dataset and simulation model, and then present the experimental results.

4.1 Dataset

The California Irrigation Management Information System (CIMIS) manages a network of over 120 automated weather stations in California [7]. Each weather station collects data in every minute and calculates hourly and daily values. The data are analyzed and stored in the CIMIS database and publicly available. The measured attributes are solar radiation, air temperature, relative humidity, wind speed, soil temperature, etc. For our experiments, we use the daily soil temperature data collected from 1998 to 2009, and implanted the random synthesized outliers in them along with inherent outliers. We use fifty stations in random and report the average results. On average each station has 4000 rounds of data (total 200,000 data rounds).The first 500 data points are used for bootstrapping from each data stream. This dataset has consecutive rounds of inherent outliers. We use 7% outliers for all of our experiments, except for those experiments in which we vary the percentage of outliers to study its impacts on the algorithms' performance.

4.2 Simulation Model

In our simulation model we mimic the typical data streams architecture. Each data source produces one data stream. We create the virtual data sources and the virtual base station. Each virtual data source obtains a data value at a fixed interval and sends it to the virtual base station. The virtual base station receives one data point from one data stream at a time and processes it. We execute DBOD-DS, ART and ODTS, one technique at a time, at the base virtual base station to detect the outliers. The entire simulation model is built on the Java platform and we ran the simulation using GNU Compiler for the Java version 1.4.2. The GNU was running on Red Hat Linux Enterprise 5 [29]. We use the Cluster Supercomputer at the University of Oklahoma to run our simulation experiments. The comparison is fair since each technique is run on the same machine.

4.3 Accuracy

We measure the accuracy in terms of Jaccard Coefficient (JC) and Area Under the receiver operator characteristic Curve (AUC). A good outlier detection technique is the one which maximizes true positive (TP) and minimizes false negative (FN) and false positive (FP). Basu and Meckesheimer [3] proposed the use of Jaccard Coefficient (JC) as a performance metric for outlier detection. Mathematically JC defined as $JC = \frac{TP}{FP+FN+TP}$. The metric (JC) consider the true positive, false negative and false positive. JC is inversely proportional to the wrong classification and directly proportional to the correct classification, and assigns equal weight to the correct classification and the wrong classification [3]. So, the better JC an outlier detection algorithm yields the more accurate results the algorithm provides. However, JC is not independent of the distribution of inliers and outliers. We use the receiver operator characteristic (ROC) curve to establish a distribution independent performance metric. On top of this, ROC curve has two other fascinating properties: 1) the ROC curve is not sensitive to a particular choice of the cut-off value and 2) the Area Under the ROC curve (AUC) provides a single scalar value which represents the performance of the classifier [12]. The ROC curve is a two dimensional graph in which the true positive rate (TPR, $TPR = \frac{TP}{TP+FN}$) goes along the y-axis and the false positive rate (FPR, $FPR = \frac{FP}{FP+TN}$, TN is true negative) goes along the x-axis. TPR is the rate of correct classification (called benefit) and FPR is the wrong classification (called cost); hence the ROC curve is the graph of cost vs. benefit [12]. The algorithm which has higher AUC is considered as a better algorithm. The optimal algorithm will increase the TPR without increasing the FPR; if we push it further it will increase the FPR only because there is no room for improvement of TPR; hence the graph will be two line segments joining $(0,0)$ to $(0,1)$ which is called conservative region and $(0,1)$ to $(1,1)$ which is called flexible region. So the better algorithm will follow the curve of the optimal algorithm. The results for ART, ODTS and DBOD-DS are reported for the optimal cut-off value which maximizes the Jaccard Coefficient of the respective algorithms. The next two sections (4.3.1 and 4.3.2) compare the three algorithms in terms of JC and ROC, respectively.

4.3.1 Jaccard Coefficient (JC)

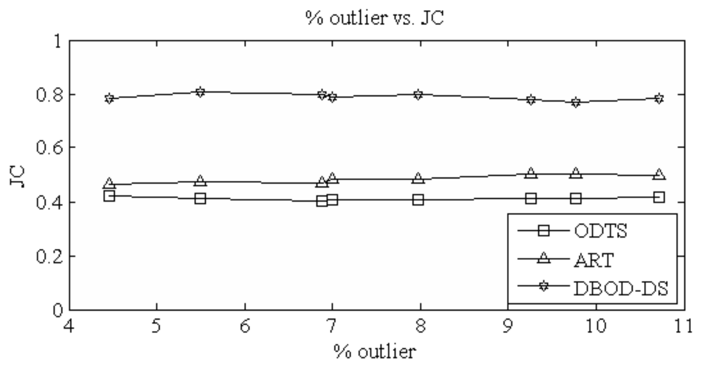

Fig 3. JC of each algorithm

Figure 3 shows the Jaccard Coefficient with respect to different percentages of outliers for our dataset. DBOD-DS outperforms all other algorithms regardless of the percentage of outliers. The JC for DBOD-DS is almost twice of the JC of the other two algorithms. DBOD-DS, ART and ODTS show constant JC with respect to change of the percentage of outliers. If the percentage of outlier increases, the true positive increases along with false negative and false positive; hence the increment of the numerator and denumerator in the JC formula makes JC constant with respect to the percentage of outliers.

4.3.2 Receiver operator characteristic curve

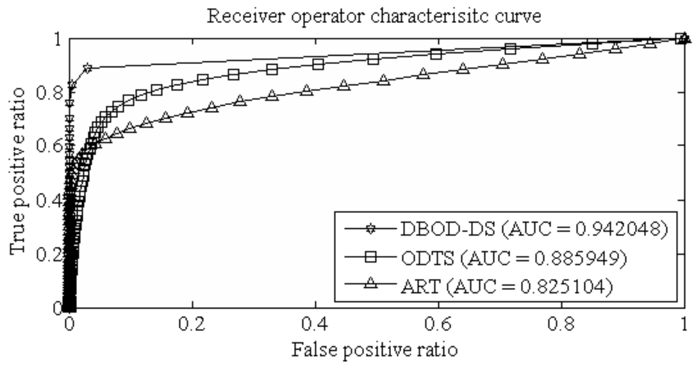

Fig 4. Receiver operator characteristic curve.

Mostly the performance of the ART, ODTS and DBOD-DS depends on the correctness of their respective thresholds. Hence, we compare DBOD-DS with those two algorithms to establish a parameter less comparison metric. Figure 4 shows the ROC curve for DBOD-DS, ART and ODTS for the dataset. DBOD-DS performs very well in the conservative region [12]; it correctly identifies the outliers without increasing the false positive ratio after the true positive ratio reaches 0.8, which is very close to the optimal performance. The optimal performance in the conservative region resembles the fact that DBOD-DS is capable of identifying true positives without increasing false positives (the sharp transition from the conservative region to the flexible region in Figure 4 confirms this fact). The most important plus point for the ROC curve is that the area under the curve (AUC) resembles a single metric for performance comparison among two classifiers. The AUC for DBOD-DS is 0.94 and the AUC for ART and ODTS are 0.82 and 0.88, respectively.. Interestingly, the performance of ODTS is better than that of ART in terms of AUC. This is because ODTS produces fewer false negatives than ART. The most appealing characteristic of AUC is that it resembles the probability of correct classification regardless of the percentage of outliers; therefore, in terms of AUC, DBOD-DS is much more superior to ART and ODTS.

4.4 Execution Time

The DBOD-DS performs much better than the other two algorithms in terms of JC and AUC, but this performance benefit does not come without cost. The DBOD-DS takes more execution time compared to ART and ODTS. Figure 5 shows the execution time for the algorithms with respect to the change of the percentage of outliers. The time is recorded for each round from receiving a data point to identifying its outlier-ness. On an average DBOD-DS takes twice more time than ART and 20 times more time than ODTS but the execution time for DBOD-DS is less than 1.5 milliseconds. The outlier detection takes place within two rounds and this time is practically enough for any type of data stream. In a typical data stream application the data source is kept onsite and the data values travel from the data source to the base station. Sending frequency lower than 1 millisecond is impractical for most of the current data stream applications. The execution time increases a little bit with the increase of the percentage of outlier; this is because if the percentage of outliers increases, the dispersion of the probability density function increases, hence more bin needs update for each data point. In our opinion the extra time is worthy for DBOD-DS because it offers a significant performance improvement over ART and ODTS in terms accuracy.

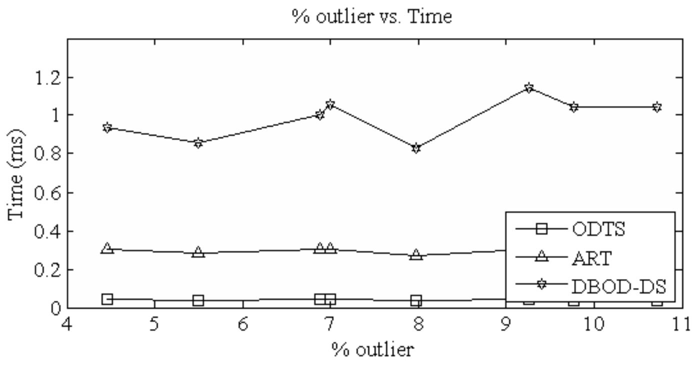

Fig 5. Execution time for each algorithm with respect to percentage of outliers

5 Conclusions and Future Research

We have developed an outlier detection algorithm for data stream applications based on our novel probability density estimation function. The performance of our algorithm compared with that of the existing algorithms in the literature is shown by extensive empirical studies on a real dataset. Our algorithm outperforms the existing algorithms in terms of accuracy, but requires more time to execute. However, the time our algorithm needs is less than 1.5 milliseconds, which is much smaller than the time required sending and receiving data in many data stream applications. From our empirical studies it is clear that our algorithm can perform excellently for a reasonable percentage of outliers. Even though we designed the algorithm considering both single dimensional data and multi dimensional data, our experiments so far have focused on the former case. In our future experiments we want to cover multi dimensional data. In addition, we want to extend our novel probability density function to estimate the data values and to detect concept drifts. In our technique we require user-defined parameters to identify outliers; it would be interesting to make our approach completely intelligent so that it would not expect any parameter from the user.

6 References

1. Anguiulli F. and Fassetti F.: Detecting Distance-Based Outliers in Streams of Data In Proceedings of the sixteenth ACM conference on Conference on information and knowledge management: Pages 811 – 820 (2007)
2. Babcock B., Babu S., Mayur D., Motwani R., Widom J.: Models and Issues in Data Stream Systems: In proceedings of 21st ACM Symposium on Principles of Database Systems (PODS 2002). Pages: 1 – 16 (2002).
3. Basu S. and Meckesheimer M.: Automatic outlier detection for time series: an application to sensor data: Knowledge Information System, pages: 137 – 154 (2007).
4. Barnett V. and Lewis T.: Outliers in Statistical Data: Wiley Series in Probability and Mathematical Statistics, Publisher: John Wiley & Sons Inc (1994).
5. Carney D., Çetintemel U., Cherniack M., Convey C., Sangdon Lee, Seidman G., Stonebraker M., Tatbul N., and Zdonik S.: Monitoring Streams – A New Class of Data Management Applications: In proceedings of the 28th international conference on Very Large Data Bases. Pages 215 – 226 (2002).
6. Chandola V., Banarjee A. and Kumar V.: Outlier Detection : A Survey: Technical Report, University of Minnesota (2007).
7. California Irrigation Management Information System, web-link: http://wwwcimis.water.ca.gov/cimis/welcome.jsp, (accessed January, 2010).
8. Curiac D., Banias O., Dragan F., Volosencu C., and Dranga O.: Malicious Node Detection in Wireless Sensor Networks Using an Autoregression Technique: In proceedings of the Third International Conference on Networking and Services, Pages 83 – 88 (2007).
9. Eskin E.: Anomaly Detection over Noisy Data using Learned Probability Distributions: In proceedings of the Seventeenth International Conference on Machine Learning. Pages 255 – 262 (2000).
10. Franke C. and Gertz M.: Detection and Exploration of Outlier Regions in Sensor Data Streams: In IEEE International Conference on Data Mining Workshop, Pages 375 – 384 (2008).
11. Franke C. and Gertz M.: ORDEN: outlier region detection and exploration in sensor networks: In proceedings of the 35th SIGMOD international conference on Management of data, Year 2009, Pages 1075 – 1078.
12. Fawcett T.: Roc graphs: Notes and practical considerations for data mining researchers: Technical report hpl-2003-4, HP Laboratories, Palo Alto, CA, USA, (2003).
13. Fan J. and James S. J. S.: Fast Implementations of Nonparametric Curve Estimators: Journal of Computational and Graphical Statistics, Vol. 3, No. 1, Pages 35-56 (1994).
14. Gilbert A. C., Kotidis Y., Muthukrishnan S., and Strauss M.: Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries: In Proceedings of the 27th International Conference on Very Large Databases, Pages 79 – 88 (2001).
15. Guha S. and Koudas N.: Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation: In Proceedings 18th International Conference on Data Engineering, Pages 567 – 676 (2002).
16. Ishida K. and Kitagawa H.: Detecting Current Outliers: Continuous Outlier Detection over Time-Series Data Streams: Lecture Notes in Computer Science, Spinger Publications, Volume 5181, Pages 255 – 268, (2008).

17. Jiang N. and Gruenwald L.: Research issues in Data Stream Association Rule Mining. ACM SIGMOD RECORD, Volume 35 , Issue 1. Pages: 14 – 19 (2006).
18. Keogh E., Lin J. and Truppel W.: Clustering of Time Series in Meaningless: Implications for Previous and Future Research: In proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, Pages 56 – 65, (2003).
19. M. Knorr E. M. and Ng R. T.: Algorithms for Mining Distance-Based Outliers in Large Datasets: In proceedings of the 24rd International Conference on Very Large Data Bases, Pages 392 – 403 (1998).
20. Madsen H.: Time Series Analysis: Texts in Statistical Science. Publisher: Chapman & Hall/CRC (2007).
21. Ng R. T. and Han J.: Efficient and Effective Clustering Methods for Spatial Data Mining: In proceedings of the 20th International Conference on Very Large Data Bases, Pages 144 – 155 (1994).
22. Puttagunta V. and Kalpakis K.: Adaptive Methods for Activity Monitoring of Streaming Data: In proceedings of International Conference on Machine Learning and Applications, Pages 197-203 (2002).
23. Scott D. W.: Multivariate Density Estimation: A Wiley-Interscience Publication (1992).
24. Sheikholeslami G., Chatterjee S., and Zhang A.: WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases: In proceedings of the 24rd International Conference on Very Large Data Bases. Pages 428 – 439 (1998).
25. Stonebraker M., Çetintemel U., Zdonik S.: The 8 Requirements of Real-Time Stream Processing: ACM SIGMOD Record Volume 34, Issue 4. Pages: 42-47 (2005).
26. Subramaniam S., Palpanas T., Papadoppoulos D., Kalogeraki V. and Gunopulos D.: Online Outlier Detection in Sensor Data Using Non-Parametric Models: In proceedings of the 32th international conference on VLDB, Pages 187-198 (2006).
27. Gruenwald L., Chok H., Aboukhamis M.: Using Data Mining to Estimate Missing Sensor Data: In Seventh IEEE International Conference on Data Mining Workshops, Pages 207 – 212 (2007).
28. Brailsford T. J., Penm J. H. W., and Terrell R. D.: Selecting the forgetting factor in Subset Autoregressive Modelling: In Journal of Time Series Analysis, Vol 23, Pages 629 – 650 (2002).
29. OU Supercomputer Resources: web-link http://www.oscer.ou.edu/resources.php (accessed May, 2010).