# Detecting Distance-Based Outliers in Streams of Data

Fabrizio Angiulli
DEIS, Università della Calabria
Via P. Bucci, 41C
87036 Rende (CS), Italy
f.angiulli@deis.unical.it

Fabio Fassetti
DEIS, Università della Calabria
Via P. Bucci, 41C
87036 Rende (CS), Italy
ffassetti@deis.unical.it

## ABSTRACT

In this work a method for detecting distance-based outliers in data streams is presented. We deal with the sliding window model, where outlier queries are performed in order to detect anomalies in the current window. Two algorithms are presented. The first one exactly answers outlier queries, but has larger space requirements. The second algorithm is directly derived from the exact one, has limited memory requirements and returns an approximate answer based on accurate estimations with a statistical guarantee. Several experiments have been accomplished, confirming the effectiveness of the proposed approach and the high quality of approximate solutions.

## Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*

## General Terms

Algorithms, Performance

## Keywords

Anomaly detection, data streams, distance-based outliers

## 1. INTRODUCTION

In many emerging applications, such as fraud detection, network flow monitoring, telecommunications, data management, etc., data arrive continuously, and it is either unnecessary or impractical to store all incoming objects. In this context, an important challenge is to find the most exceptional objects in the stream of data.

A *data stream* is a large volume of data coming as an unbounded sequence, where, typically, older data objects are less significant than more recent ones, and thus should contribute less. This is because characteristics of the data may change during the evolution, and then the most recent behavior should be given larger weight.


*CIKM'07,* November 6–8, 2007, Lisboa, Portugal.


Therefore, data mining on evolving data streams is often performed based on certain time intervals, called windows. Two main different data streams window models have been introduced in literature: *landmark window* and *sliding window* [15].

In the first model, some time points (called landmarks) are identified in the data stream, and analysis are performed only for the stream portion which falls between the last landmark and the current time. Then, the window is identified by a fixed endpoint and a moving endpoint.

In contrast, in the sliding window model, the window is identified by two sliding endpoints. In this approach, old data points are thrown out as new points arrive. In particular, a *decay function* is defined, that determines the weight of each point as a function of elapsed time since the point was observed. A meaningful decay function is the step function. Let $W$ be a parameter defining the window size and let $t$ denote the current time, the step function evaluates to 1 in the temporal interval $[t - W + 1, t]$, and 0 elsewhere.

In all window models the main task is to analyze the portion of the stream within the current window, in order to mine data stream properties or to single out objects conforming with characteristics of interest.

In this work, the problem of detecting objects, called *outliers*, that are abnormal with respect to data within the current window, is addressed. Specifically, the sliding window model with the step function as decay function is adopted.

There exist several approaches for the problem of singling out the objects mostly deviating from a given collection of data [8, 6, 17, 11, 16, 1, 22]. In particular, distance-based approaches [17] exploit the availability of a distance function relating each pair of objects of the collection. They identify as outliers the objects lying in the most sparse regions of the feature space.

Distance-based definitions [17, 24, 4] represent an useful tool for data analysis [18, 14, 21]. Given parameters $k$ and $R$, an object is a distance-based outlier if less than $k$ objects in the input data set lie within distance $R$ from it [17]. Distance-based outliers have robust theoretical foundations, since they are a generalization of different statistical tests. Furthermore, they are computationally efficient, since distance-based outlier scores are a monotonic non-increasing function of the portion of the database already explored.

Two algorithms for detecting distance-based outliers in data streams are presented. The first one exactly answers outlier queries at any time, but has larger space requirements. The second algorithm is directly derived from the exact one, has limited memory requirements and returns an

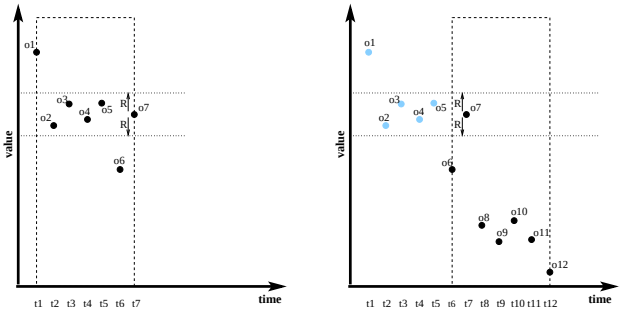
approximate answer based on highly accurate estimations with a statistical guarantee.

The approach proposed in this work introduces a novel concept of *querying* for outliers. Specifically, previous work deals with *continuous queries*, that are queries evaluated continuously as data stream objects arrive; conversely, we deal with *one-time query*, that are queries evaluated once over a point-in-time (for a survey on data streams query models refer to [7]).

The underlying intuition is that, due to stream evolution, object properties can change over time and, hence, evaluating an object for outlierness when it arrives, although meaningful, can be reductive in some contexts and sometimes misleading. On the contrary, by classifying single objects when a data analysis is required, data *concept drift* typical of streams can be captured. To this aim, it is needed to support queries at arbitrary points-in-time, called *query times*, which classify the whole population in the current window instead of the single incoming data stream object. To our best knowledge this is the first work performing outlier detection on windows at query time.

The example below shows how concept drift can affect the outlierness of data stream objects.

*Example 1.* Consider the following figure.




The two diagrams represent the evolution of a data stream of *1-dimensional* objects. The abscissa reports the time of arrival of the objects, while the ordinate reports the value assumed by each object. Let the number $k$ of nearest neighbors to consider be equal to 3, and let the window size $W$ be equal to 7. The dashed line represents the current window.

The left diagram reports the current window at time $t_7$ (comprehending the interval $[t_1, t_7]$), whereas the right diagram reports the current window at time $t_{12}$ (comprehending the interval $[t_6, t_{12}]$).

First of all, consider the diagram on the left. Due to the data distribution in the current window at time $t_7$, the object $o_7$ is an inlier, since it has four neighbors in the window. Then, if an analysis were required at time $t_7$, the object $o_7$ would be recognized as an inlier. Note that $o_7$ belongs to a very dense region.

Nevertheless, when stream evolves the data distribution changes. The region, which $o_7$ belongs to, becomes sparse, and data stream objects assume lower values. On the right figure is reported the evolution of the stream until time $t_{12}$. In the novel distribution, $o_7$ has not any neighbor. Then, if an analysis were required at time instant $t_{12}$, $o_7$ should be recognized as an outlier. Note that now $o_7$ belongs to a very sparse region. □

The contribution of this work can be summarized as follows:

- the novel task of data stream outlier query is introduced;
- an exact algorithm to efficiently detect distance-based outliers in the introduced model is presented;
- an approximate algorithm is derived from the exact one, based on a trade off between spatial requirements and answer accuracy; the method approximates object outlierness with a statistical guarantee;
- by means of experiments on both real and synthetic data sets, the efficiency and the accuracy of the proposed techniques are shown.

The rest of the work is organized as follows. Section 2 briefly surveys related outlier detection approaches and data stream algorithms. Subsequent Section 3 formally states the data stream outlier query problem which this work deals with. Section 4 describes both the exact and the approximate algorithm. Section 5 illustrates experimental results. Finally, Section 6 presents conclusions.

## 2. RELATED WORK

*Distance-based outliers* have been first introduced by Knorr and Ng [17]. Given parameters $k$ and $R$, an object is a distance-based outlier if less than $k$ objects in the input data set lie within distance $R$ from it. This definition is a solid one, since it generalizes several statistical outlier tests.

Some variants of the original definition have been subsequently introduced in literature. In particular, Ramaswamy et al. [24], in order to rank the outliers, introduced the following definition: given $k$ and $n$, an object $o$ is an outlier if no more than $n-1$ other objects in the dataset have higher value for $D^k$ than $o$, where $D^k(o)$ denotes the distance of the $k$th nearest neighbor of $o$. Subsequently, Angiulli and Pizzuti [4, 5, 3], with the aim of taking into account the whole neighborhood of the objects, proposed to rank them on the basis of the sum of the distances from the $k$ nearest neighbors, rather than considering solely the distance to the $k$th nearest neighbor. In this work we will deal with the definition provided in [17].

Knorr et al. [17, 19] presented two algorithms. The first one is a block nested loop algorithm that runs in $O(dN^2)$ time, where $N$ is the number of points and $d$ the number of dimensions of the data set, while the second one is a cell-based algorithm that is linear with respect to $N$, but exponential in $d$. The last method is fast only if $d$ is small. On the other hand, the nested loop approach does not scale well. Thus, efforts for developing efficient algorithms that scale to large datasets have been subsequently made.

In [24] the authors presented two novel algorithms to detect outliers. The first assumes the dataset is stored in a spatial index, like the $R^*$-tree [10]. The second algorithm, first partitions the input points using a clustering algorithm, and then prunes the partitions that cannot contain outliers.

Bay and Schwabacher [9], introduced the distance-based outlier detection algorithm ORCA. Basically, ORCA enhances the naive block nested loop algorithm with a simple pruning rule and randomization, obtaining a near linear scaling on large and high dimensional data sets. The CPU time of the this nested loop schema is often approximately linear in the dataset size.

Distance-based methods previously discussed are designed to work in the batch framework, that is under the assumption that the whole data set is stored in secondary memory and multiple passes over the data can be accomplished.

Hence, they are not suitable for data streams. While the majority of the approaches to detect anomalies in data mining consider the batch framework, some researchers have attempted to address the problem of online outlier detection.

In [27] the SmartSifter system is presented, addressing the problem from the viewpoint of statistical learning theory. The system employs an online discounting learning algorithm to learn a probabilistic model representing the data source. Every time a datum is input, SmartSifter evaluates how large the datum is deviated relative to a normal pattern. In [2] the focus is on detecting rare events in a data stream. Rare events are defined on the basis of their deviation from expected values computed on historical trends. In [25] a distributed framework to approximate data distributions coming from a sensor network is presented. Kernel estimators are exploited to approximate data distributions. The technique is used to report outlying values in the union of the readings coming from multiple sensors.

The last work applies the outlier technique presented with two outlier definitions, i.e. distance-based and MDEF-based [23]. However, it must be said that this method is specifically designed to support sensor networks. Moreover, all the techniques above discussed, detect anomalies online as they arrive, and one-time queries are not supported.

# 3. STATEMENT OF THE PROBLEM

Next, the formalism employed in this work will be presented. First of all, the formal definition of distance-based outlier is recalled [17].

DEFINITION 3.1 (DISTANCE-BASED OUTLIER). *Let $S$ be a set of objects,* obj *an object of $S$, $k$ a positive integer, and $R$ a positive real number. Then,* obj *is a* distance-based outlier *(or, simply, an* outlier*) if less than $k$ objects in $S$ lie within distance $R$ from* obj.

Objects lying at distance not greater than $R$ from *obj* are called *neighbors* of *obj*. The object *obj* is not considered a neighbor of itself.

A *data stream* **DS** is a possible infinite series of objects $\ldots, obj_{t-2}, obj_{t-1}, obj_t, \ldots$, where $obj_t$ denotes the object observed at time $t$. We will interchangeably use the term *identifier* and the term *time of arrival* to refer to the time $t$ at which the object $obj_t$ was observed for the first time.

We assume that data stream objects are elements of a metric space on which is defined a distance function. In the following we will use $d$ to denote the space required to store an object, and $\Delta$ to denote the temporal cost required for computing the distance between two objects.

Given two object identifiers $t_m$ and $t_n$, with $t_m \leq t_n$, the *window* $\mathbf{DS}[t_m, t_n]$, is the set of $n - m + 1$ objects $obj_{t_m}$, $obj_{t_m+1}$, ..., $obj_{t_n}$. The *size* of the window $\mathbf{DS}[t_m, t_n]$ is $n - m + 1$.

Given a window size $W$, the *current window* is the window $\mathbf{DS}[t - W + 1, t]$, where $t$ is the time of arrival of the last observed data stream object.

An *expired object obj* is an object whose identifier *id* is less than the lower limit of the current window, i.e. such that $id < t - W + 1$.

Now, we are in the position of defining the main problem we are interested in solving.

DEFINITION 3.2 (DATA STREAM OUTLIER QUERY). *Given a data stream* **DS***, a window size $W$, and fixed parameters $R$ and $k$, the* Data Stream Outlier Query *is: return the distance based outliers in the current window.*

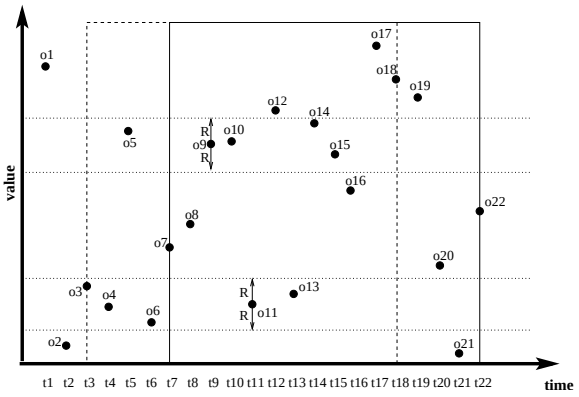
The time $t$, which a data stream outlier query is requested at, is called *query time*.

In the following the neighbors of an object *obj* that precede *obj* in the stream and belong to the current window are called *preceding neighbors* of *obj*. Furthermore, the neighbors of an object *obj* that follow *obj* in the stream and belong to the current window are called *succeeding neighbors* of *obj*.

According to Definition 3.1, an *inlier* is an object *obj* having at least $k$ neighbors in the current window. In other words, let $\alpha$ be the number of preceding neighbors of *obj* and $\beta$ be the number of succeeding neighbors of *obj*, *obj* is an inlier if $\alpha + \beta \geq k$.

Let *obj* be an inlier. Since during stream evolution objects expire, the number of preceding neighbors of *obj* decreases. Therefore, if the number of succeeding neighbors of *obj* is less than $k$, *obj* could become an outlier depending on the stream evolution. Conversely, since *obj* will expire before its succeeding neighbors, inliers having at least $k$ succeeding neighbors will be inliers for any stream evolution. Such inliers are called *safe inliers*.

*Example 2.* The following diagram represents the evolution of a one-dimensional data stream. Let $k$ be 3, and let $W$ be 16.




Consider the current window (the dashed one) at time $t_{18}$: both $o_9$ and $o_{11}$ are inliers, since $o_9$ has four neighbors ($o_5, o_{10}, o_{14}, o_{15}$), and also $o_{11}$ has four neighbors ($o_3, o_4, o_6, o_{13}$). Moreover, since $o_9$ has three succeeding neighbors, it is a safe inlier, while $o_{11}$ is not a safe inlier.

Indeed, consider instant $t_{22}$. The object $o_9$ is still an inlier: object $o_5$ expired, but $o_9$ has still three (succeeding) neighbors. Conversely, $o_{11}$ is now an outlier: objects $o_3$, $o_4$ and $o_6$ expired, and now it has only one neighbor. □

# 4. ALGORITHM

In this section the algorithm STORM, standing for STream OutlieR Miner, is described. Two variants of the method are presented.

When the entire window can be allocated in memory, the exact answer of the data stream outlier query can be computed. In the above setting, the algorithm exact-STORM is able to exactly answer outlier queries at any time (Section 4.1).

Nevertheless, interesting windows are often so large that they do not fit in memory, while in some applications only

limited memory can be allocated . In this case approximations must be employed. The algorithm approx-STORM is designed to work in the latter setting by introducing effective approximations in exact-STORM (Section 4.2). These approximations guarantee highly accurate answers with limited memory requirements (Section 4.3).

After having described the two methods, temporal and spatial costs required to obtain exact and approximate answers will be stated (Section 4.4).

## 4.1 Exact algorithm

The algorithm exact-STORM is shown in Figure 1. This algorithm consists of two procedures: the *Stream Manager* and the *Query Manager*. The former procedure receives the incoming data stream objects and efficiently updates a suitable data structure that will be exploited by the latter procedure to effectively answer queries.

In order to maintain a summary of the current window, a data structure, called ISB (standing for *Indexed Stream Buffer*), storing *nodes* is employed (nodes are defined next). Each node is associated with a different data stream object.

ISB provides a method *range query search*, that, given an object *obj* and a real number $R \geq 0$, returns the nodes in ISB associated with objects whose distance from *obj* is not greater than $R$. We will describe the implementation of ISB in Section 4.4.

Now, the definition of node is provided. A *node* $n$ is a record consisting of the following information:

- $n.obj$: a data stream object.
- $n.id$: the identifier of $n.obj$, that is the arrival time of $n.obj$.
- $n.count\_after$: the number of succeeding neighbors of $n.obj$. This field is exploited to recognize safe inliers.
- $n.nn\_before$: a list, having size at most $k$, containing the identifiers of the most recent preceding neighbors of $n.obj$. At query time, this list is exploited to recognize the number of preceding neighbors of $n.obj$. We assume that both the operation of *ordered insertion* of a novel identifier in the list and the operation of *search* of an identifier in the list are executed in time $\mathcal{O}(\log k)$ (see [20] for a suitable implementation).

The *Stream Manager* takes as input a data stream **DS**, a window size $W$, a radius $R$, and the number $k$ of nearest neighbors to consider.

For each incoming data stream object *obj*, a novel node $n_{curr}$ is created with $n_{curr}.obj = obj$. Then a range query search with center $n_{curr}.obj$ and radius $R$ is performed in ISB, that returns the nodes associated with the preceding neighbors of *obj* stored in ISB.

For each node $n_{index}$ returned by the range query search, since the object *obj* is a succeeding neighbor of $n_{index}.obj$, the counter $n_{index}.count\_after$ is incremented. Moreover, since the object $n_{index}.obj$ is a preceding neighbor of *obj*, the list $n_{curr}.nn\_before$ is updated with $n_{index}.id$.

If the counter $n_{index}.count\_after$ becomes equal to $k$, then the object $n_{index}.obj$ becomes a safe inlier. Thus, it will not belong to the answer of any future outlier query. Despite this important property, a safe inlier cannot be discarded from ISB, since it may be a preceding neighbor of a future stream object. Finally, the node $n_{curr}$ is inserted into ISB. This terminates the description of the procedure *Stream Manager*.

In order to efficiently answer queries, when invoked by the user, the *Query Manager* performs a single scan of ISB. In particular, for each node $n$ of ISB, the number *prec_neighs* of identifiers stored in $n.nn\_before$ associated with non-expired objects is determined. This is accomplished in $\mathcal{O}(\log k)$ time by performing a search in the list $n.nn\_before$ of the identifier closest to the value $t - W + 1$, that is the identifier of the oldest object in $n.nn\_before$ belonging to the current window.

The number *succ_neighs* of succeeding neighbors of $n.obj$ is stored in *count_after*. Thus, if *prec_neighs* + *succ_neighs* $\geq k$ then the object $n.obj$ is recognized as an inlier, otherwise it is an outlier and is included in the answer of the outlier query.

## 4.2 Approximate algorithm

Figure 2 shows the algorithm approx-STORM. The exact algorithm requires to store all the window objects. If the window is so huge that does not fit in memory, or only limited memory can be allocated, the exact algorithm could be not employable. However, as described in the following, the algorithm described in the previous section can be readily modified to reduce the space required.

With this aim, two approximations are introduced.

Firstly, in order to severely reduce the space occupied, we do not store all the window objects into ISB. In particular, objects belonging to ISB can be partitioned in outliers and inliers. Among the latter kind of objects there are safe inliers, that are objects that will be inliers in any future stream evolution. As already observed, despite safe inliers cannot be returned by any future outlier query, they have to be kept in ISB in order to correctly recognize outliers, since they may be preceding neighbors of future incoming objects.

However, as shown in subsequent Section 4.3, it is sufficient to retain in ISB only a fraction of safe inliers to guarantee an highly accurate answer to the outlier query. Thus, in order to maintain in ISB a controlled fraction $\rho$ ($0 \leq \rho \leq 1$) of safe inliers, the following strategy is adopted.

During stream evolution, an object *obj* of the stream becomes a safe inlier when exactly $k$ succeeding neighbors of *obj* arrive. At that time, if the total number of safe inliers into ISB exceeds $\rho W$, then a randomly selected object of ISB is removed. The random selection policy adopted guarantees that safe inliers surviving into ISB are uniformly distributed in the window.

To answer one-time queries both outliers and non-safe inliers, which are objects candidate to become outliers if the stream characteristics change, have to be maintained in ISB. Note that, meaningful combinations of the parameters $R$ and $k$ are only those for which the number of outliers, and hence of non-safe inliers, amounts to a negligible fraction of the overall population. Thus, the number of nodes in ISB can be assumed approximately equal to $\rho W$. In the following section it will be discussed how to compute an optimal value for $\rho$ in order to obtain a statistical guarantee on the approximation error of the estimation of the number of preceding neighbors of each data stream object.

The second approximation consists in reducing the size of each node by avoiding storing the list of the $k$ most recent preceding neighbors. This is accomplished by storing

**Procedure** *Stream Manager*
**Input:** **DS** is the data stream;
$W$ is the window size;
$R$ is the neighborhood radius;
$k$ is the number of neighbors.
**Method:**

For each data stream object *obj* with identifier $t$:

1. if the oldest node $n_{oldest}$ of ISB expires, then remove the node $n_{oldest}$ from ISB;
2. create a new node $n_{curr}$, with $n_{curr}.obj = obj$, $n_{curr}.id = t$, $n_{curr}.nn\_before = \emptyset$, $n_{curr}.count\_after = 1$;
3. perform a range query search with center *obj* and radius $R$ into ISB. For each node $n_{index}$ returned by the range query:
   (a) increment the value $n_{index}.count\_after$;
   (b) update the list $n_{curr}.nn\_before$ with the object identifier $n_{index}.id$;
4. insert the node $n_{curr}$ into ISB.

**Procedure** *Query Manager*
**Output:** the distance-based outliers in the current window;
**Method:**

1. For each node $n$ stored in ISB:
   (a) let *prec_neighs* be the number of identifiers stored in $n.nn\_before$ associated with non-expired objects, and let *succ_neighs* be $n.count\_after$;
   (b) if $prec\_neighs + n.succ\_neighs \geq k$ then mark $n.obj$ as inlier, else mark it as an outlier;
2. return all the objects marked as outliers.

**Figure 1: Exact data stream distance-based outlier detection algorithm.**

in each node $n$, instead of the list $n.nn\_before$, just the fraction $n.fract\_before$ of previous neighbors of $n.obj$ observed in ISB at the arrival time $n.id$ of the object $n.obj$. The value $n.fract\_before$ is determined as the ratio between the number of preceding neighbors of $n.obj$ in ISB which are safe inliers and the total number of safe inliers in ISB, at the arrival time of $n.obj$.

At query time, in order to recognize outliers, a scan of ISB is performed, and, for each stored node $n$, the number of neighbors of $n.obj$ in the current window has to be evaluated. Since only the fraction $n.fract\_before$ is stored now in $n$, the number of preceding neighbors of $n.obj$ in the whole window at the current time $t$ has to be estimated.

Let $\alpha$ be the number of preceding neighbors of $n.obj$ at the arrival time of $n.obj$. Assuming that they are uniformly distributed along the window, the number of preceding neighbors of $n.obj$ at the query time $t$ can be estimated as

$$prec\_neighs = \alpha \cdot \frac{W - t + n.id}{W}.$$

Note that n.*fract_before* does not give directly the value $\alpha$, since it is computed by considering only the objects stored in ISB and, thus, it does not take into account removed safe inliers preceding neighbors of $n.obj$. However, $\alpha$ can be safely (see next section) estimated as

$$\alpha \approx n.fract\_before \cdot W.$$

Summarizing, the number of preceding neighbors of $n.obj$ at the query time $t$ can be estimated as

$$prec\_neighs = n.fract\_before \cdot (W - t + n.id).$$

Recall that to classify objects the sum between the estimated number of its preceding neighbors and the number of succeeding neighbors is computed. It is worth to point out that the number of succeeding neighbors is not estimated, since $n.count\_before$ provides the *true* number of succeeding neighbors of $n.obj$. Therefore as stream evolves, the above sum approaches the true number of neighbors in the window.

## 4.3 Approximation Error Bounds

Now, we discuss how to set the parameter $\rho$ in order to obtain safe bounds on the approximation error of the estimation.

Let $W$ be the window size, let $w$ be the number of safe inliers in ISB, let $\alpha$ be the exact number of preceding neighbors of an object at its time of arrival, let $\widetilde{\alpha}$ be the number of preceding neighbors of the object in ISB which are safe inliers at its time of arrival, and let $\mu$ denote the ratio $\frac{\alpha}{W}$.

In order to determine an optimal value for $\rho$, a value for $w$, such that $\frac{\widetilde{\alpha}}{w}$ is a good approximation for $\mu$, has to be determined. Formally, the following property has to be satisfied. *For given $\delta > 0$ and $0 < \epsilon < 1$, we want to have:*

$$\Pr\left[\left|\frac{\widetilde{\alpha}}{w} - \mu\right| \leq \epsilon\right] > 1 - \delta. \quad (1)$$

Since ISB contains a random sample of the window safe inliers, a safe bound for $w$ can be obtained from the well known *Lyapounov Central Limit Theorem.*

This theorem asserts that, for any $\lambda$

$$\lim_{w \to \infty} Pr\left[\frac{\widetilde{\alpha} - w\mu}{\sqrt{w\mu(1-\mu)}} \leq \lambda\right] = \Phi(\lambda)$$

where $\Phi(\lambda)$ denotes the cumulative distribution function of the normal distribution.

**Procedure** *Stream Manager*
**Input:** $DS$ is the data stream;
$W$ is the window size;
$R$ is the neighborhood radius;
$k$ is the number of neighbors.
**Method:**

For each data stream object $obj$ with identifier $t$:

1. if the oldest node $n_{oldest}$ of ISB expires, then remove the node $n_{oldest}$ from ISB;
2. create a new node $n_{curr}$, with $n_{curr}.obj = obj$, $n_{curr}.id = t$, $n_{curr}.count\_after = 1$, and set $count\_before = 0$;
3. perform a range query search with center $obj$ and radius $R$ into ISB. For each node $n_{index}$ returned by the range query:
   (a) increment the value $n_{index}.count\_after$. If the number of safe inliers in ISB is greater than $\rho W$, then remove from ISB a randomly selected safe inlier;
   (b) increment the value $count\_before$;
4. set $n_{curr}.fract\_before = \frac{count\_before}{safe\_inliers}$, where $safe\_inliers$ is the number of safe inliers into ISB, and insert the node $n_{curr}$ into ISB.

**Procedure** *Query Manager*
**Output:** the distance-based outliers in the current window;
**Method:**

1. For each node $n$ stored in ISB:
   (a) let $prec\_neighs$ be $n.fract\_before \cdot (W - t + n.id)$, and let $succ\_neighs$ be $n.count\_after$;
   (b) if $prec\_neighs + succ\_neighs \geq k$ then mark $n.obj$ as inlier, else mark it as an outlier;
2. return all the objects marked as outliers.

**Figure 2: Approximate data stream distance-based outlier detection algorithm.**

Thus, if $w$ is large enough, then the following relation holds:

$$Pr\left[\frac{\widetilde{\alpha} - w\mu}{\sqrt{w\mu(1-\mu)}} \leq \lambda\right] \approx \Phi(\lambda).$$

Now, the result that will allow us to get the needed safe bound for $w$ can be formally presented.

THEOREM 4.1. *For any $\delta > 0$ and $0 < \epsilon < 1$, if $w$ satisfies the following inequality*

$$w > \frac{\mu(1-\mu)}{\epsilon^2}\left(\Phi^{-1}\left(1 - \frac{\delta}{2}\right)\right)^2 \tag{2}$$

*then it satisfies* (1).

Theorem 4.1 is a direct consequence of the central limit theorem (see [26] for details).

Now, the bound stated in the above theorem is examined. Although the provided bound depends on the unknown value $\alpha$, it can be safely applied by setting $\mu$ to $\frac{1}{2}$. Therefore, in order to satisfy (1), being $w = \rho W$, it is sufficient to set $\rho$ to the value

$$\rho = \frac{1}{4\epsilon^2 W}\left(\Phi^{-1}\left(1 - \frac{\delta}{2}\right)\right)^2. \tag{3}$$

It is worth to note that the bound for $w$ given by expression (2) does not depend on the window size $W$. Furthermore, since in expression (3) the unknown value $\mu$ is safely set to $\frac{1}{2}$, whenever $\mu$ is different to $\frac{1}{2}$ the property (1) is guaranteed for values of $\epsilon$ and $\delta$ better than those used to compute $w$. In particular, the two following inequalities hold:

$$Pr\left[\left|\frac{\widetilde{\alpha}}{w} - \mu\right| \leq \epsilon\right] > 1 - \delta^*, \text{ and} \tag{4}$$

$$Pr\left[\left|\frac{\widetilde{\alpha}}{w} - \mu\right| \leq \epsilon^*\right] > 1 - \delta. \tag{5}$$

In the first inequality, $\delta^*$ is obtained from the following equation:

$$\frac{1}{4\epsilon^2}\left(\Phi^{-1}\left(1 - \frac{\delta}{2}\right)\right)^2 = \frac{\mu(1-\mu)}{\epsilon^2}\left(\Phi^{-1}\left(1 - \frac{\delta^*}{2}\right)\right)^2$$

whereas, in the second one, $\epsilon^*$ is obtained from

$$\frac{1}{4\epsilon^2}\left(\Phi^{-1}\left(1 - \frac{\delta}{2}\right)\right)^2 = \frac{\mu(1-\mu)}{\epsilon^{*2}}\left(\Phi^{-1}\left(1 - \frac{\delta}{2}\right)\right)^2.$$

Note that, if the true value for $\mu$ is $\frac{1}{2}$, then $\delta^* = \delta$ and $\epsilon^* = \epsilon$.

Equation (5) can be rewritten as

$$Pr\left[\frac{\widetilde{\alpha}}{w} - \epsilon^* \leq \frac{\alpha}{W} \leq \frac{\widetilde{\alpha}}{w} + \epsilon^*\right] > 1 - \delta.$$

It follows that the maximum error $err$ we can make in computing $\alpha$ is

$$err = W\epsilon^* = W \cdot 2\epsilon\sqrt{\mu(1-\mu)}.$$

This value provides the maximum error made in estimating the total number of neighbors of an object when it arrives.

Now, we are interested in determining when the above error will cause a misclassification, i.e. when an inlier (resp., an outlier) will be estimated as an outlier (resp., an inlier).

For an object $obj$ having identifier $id$, when it arrives, the number of true preceding neighbors is $\mu W$. As stream evolves, some preceding neighbors expire, and, assuming they are uniformly distributed along the window, in the portion of the stream preceding $obj$ in the current window their number becomes $\mu(W - t + id)$.

To correctly classify $obj$, if the sum between the number $\alpha$ of preceding neighbors of $obj$ and the number $\beta$ of succeeding neighbors is greater (resp. smaller) than $k$ in the current window, then also the estimated value for $\alpha$ plus the number $\beta$ of succeeding neighbors should be greater (resp. smaller) than $k$. Formally, let $\overline{W} = W - t + id$ be the number of objects of the current window preceding the object $obj$ with identifier $id$, let $\alpha = \mu\overline{W}$ be the true value of preceding neighbors of $obj$ in the current window, let $\beta$ be the number of its succeeding neighbors, and let $2\overline{W}\epsilon\sqrt{\mu(1-\mu)}$ be the error $err$. If $\alpha + \beta$ is greater than $k$, then the following inequality should hold:

$$\mu\overline{W} - 2\overline{W}\epsilon\sqrt{\mu(1-\mu)} + \beta > k$$

Assuming that the distribution of succeeding neighbors is the same as the distribution of preceding neighbors, $\beta$ can be approximated to $\mu(W-\overline{W})$, where $(W-\overline{W})$ is the portion of the stream in the current window succeeding $obj$. Thus, for

$$\mu > \frac{kW + 2\overline{W}^2\epsilon^2 + \sqrt{(kW + 2\epsilon^2\overline{W}^2)^2 - k^2(W^2 + 4\overline{W}^2\epsilon^2)}}{W^2 + 4\overline{W}^2\epsilon^2} = \mu_{up} \tag{6}$$

an inlier is recognized with probability $(1 - \delta)$.

Analogously, if $\alpha + \beta < k$, starting from

$$\mu\overline{W} + 2\overline{W}\epsilon\sqrt{\mu(1-\mu)} + \beta < k$$

with the same assumption stated above, we obtain that for

$$\mu < \frac{kW + 2\overline{W}^2\epsilon^2 - \sqrt{(kW + 2\epsilon^2\overline{W}^2)^2 - k^2(W^2 + 4\overline{W}^2\epsilon^2)}}{W^2 + 4\overline{W}^2\epsilon^2} = \mu_{down} \tag{7}$$

an outlier is recognized with probability $(1 - \delta)$.

It can be concluded that, if an object $obj$ has more than $\mu_{up}\overline{W}$ or less than $\mu_{down}\overline{W}$ neighbors, it is correctly classified with probability $1 - \delta$.

Contrariwise, if the number of neighbors of $obj$ is in the range $[\mu_{down}\overline{W}, \mu_{up}\overline{W}]$ then we have an estimation error at most equal to $2\overline{W}\epsilon\sqrt{\mu(1-\mu)}$ that could lead to a misclassification. Both the error and the range are small and directly proportional to $\epsilon$. Moreover, they depend on the current time $t$. As $t$ increases, the error goes to zero and the range tends to be empty.

Before concluding, it is worth to recall that object classification depends also on the number of succeeding neighbors of the object, whose true value is known.

## 4.4 Implementation and Cost Analysis

In this section the implementation of ISB is detailed, and then, temporal and spatial costs of STORM are analyzed.

**Implementation details.** The ISB data structure is a *pivot-based* index [12]. It performs proximity search in any metric space making use of a certain number of objects (also called *pivots*) selected among the objects stored into the index. Distances among pivots and objects stored into the index are precalculated when a novel pivot is generated. When a range query with center $obj$ and radius $R$ is submitted to the index, the distances between the pivots and the query object $obj$ are computed. The precalculated distances are exploited to recognize the index objects lying at distance greater than $R$ from $obj$. For each index object $obj'$, if there exists a pivot $p$, such that $|\text{dist}(obj, p) - \text{dist}(obj', p)| > R$, then, by the triangular inequality, it is known that $\text{dist}(obj, obj') > R$, and $obj'$ is ignored. Otherwise, $obj'$ is marked as a candidate neighbor and if the distance $\text{dist}(obj', obj) \leq R$ then $obj'$ is returned as a true neighbor of $obj$. By using this kind of technique the range query search returns the list of neighbors of $obj$. The performances of pivot-based indexes are related to the number of pivots employed. In particular the larger is the number of pivots, the more accurate is the list of candidates, and then the lower is the number of distances computed. Nevertheless, the cost of querying and building the index increases. In this work the number of pivots used is logarithmic with respect to the index size. In order to face concept drift, the older pivot is periodically replaced with an incoming object.

Now, the temporal cost of operations to be executed on the ISB data structure is analyzed. Recall that the cost of computing the distance between two objects is $\Delta$. Assume that the number of nodes stored in ISB is $N$.

The cost of performing a range query search corresponds to the cost of computing the distances between an object and all the pivots plus the cost of determining true neighbors. Since the number of pivots we used is logarithmic in the size of the index, the former cost is $\mathcal{O}(\Delta \log N)$. As for the latter cost, let $\eta$ $(0 \leq \eta \leq 1)$ be the mean fraction of index objects marked as candidate neighbors when the radius is set to $R$. In order to determine if a candidate is a true neighbor, its distance from the query object has to be computed. Then, the cost is $\mathcal{O}(\Delta\eta N)$. Supposing that the latter cost is always greater than the former one, the total cost for the range query search is $\mathcal{O}(\Delta\eta N)$.

The cost of removing an object from the index is constant, since it practically consists in flagging as empty the entry of an array.

Finally, as for the insertion of an object $obj$ into ISB, it requires to compute the distances among $obj$ and all the pivots. However, since insertion is always performed after the range query search, these distances are already available and, then, the insertion cost is constant, too.

**Spatial analysis.** For exact-STORM, ISB stores all the $W$ objects of the current window and, for each of them, the list *nn_before* of the $k$ most recent preceding neighbors, and two integer numbers. Recall that each object requires space $d$ and each list requires space $k$.

For approx-STORM, assuming meaningful combinations of the parameters, ISB stores approximately $\rho W$ objects of the current window and, for each of them, a counter of preceding neighbors, and two integer numbers.

Summarizing, the spatial cost of the algorithm STORM is

- $\mathcal{O}(W(d + k))$ for exact-STORM, and
- $\mathcal{O}(\rho W d)$ for approx-STORM.

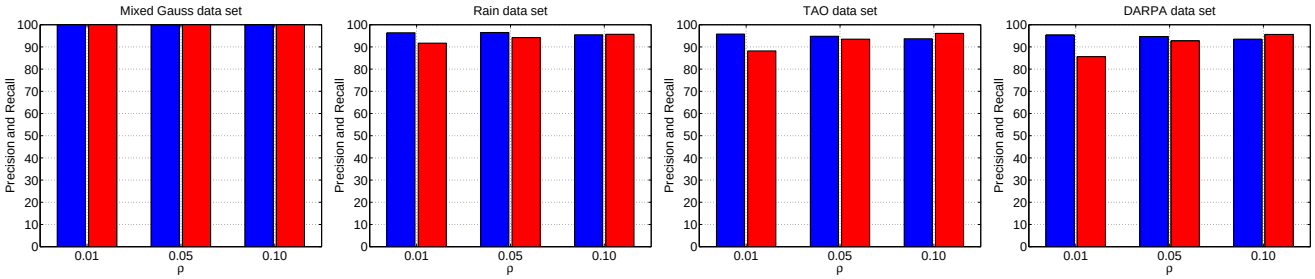



**Figure 3: Precision and Recall of approx-**STORM**.**

**Temporal analysis.** The procedure *Stream Manager* consists of four steps (see Figures 1 and 2). The cost of the step 1 corresponds to the cost of removing an object from ISB, which, from the discussion above, is constant. Also step 2 has a constant cost.

Step 3 performs a range query search, whose cost is $\mathcal{O}(\Delta\eta N)$. For each of the $\eta N$ objects returned by the search, the exact-STORM performs an ordered insertion into the list *nn_before*, that costs $\mathcal{O}(\log k)$ (see Section 4.1), and executes some other operations having constant cost. Contrariwise, the approx-STORM possibly removes a node from ISB and executes some other operations. Both the removal and the other operations have constant cost.

Finally, step 4 inserts a node into ISB and hence has constant cost. Summarizing, the cost of the procedure *Stream Manager* is

- $\mathcal{O}(\Delta\eta W \log k)$ for exact-STORM, and
- $\mathcal{O}(\Delta\eta\rho W)$ for approx-STORM.

The procedure *Query Manager* consists of a scan of the ISB structure. For each node $n$, the exact-STORM computes *prec_neighs* in time $\mathcal{O}(\log k)$ (see Section 4.1) by determining the number of identifiers in $n.nn\_before$ associated with non-expired objects. Conversely, the approx-STORM performs only steps having constant cost. Summarizing, the cost of the procedure *Query Manager* is

- $\mathcal{O}(W \log k)$ for exact-STORM, and
- $\mathcal{O}(\rho W)$ for approx-STORM.

# 5. EXPERIMENTAL RESULTS

In this section, we present results obtained by experimenting the proposed techniques on both synthetic and real data sets.

First of all, the data set employed are described. The *Gauss* data set is a synthetically generated time sequence of 35,000 one dimensional observations, also used in [23]. It consists of a mixture of three Gaussian distributions with uniform noise.

We also used some public real data from the Pacific Marine Environmental Laboratory of the U.S. National Oceanic & Atmospheric Administration (NOAA). Data consist of temporal series collected in the context of the Tropical Atmosphere Ocean project (TAO)[1]. This project collects real-time data from moored ocean buoys for improved detection, understanding and prediction of El Niño and La Niña, which are oscillations of the ocean-atmosphere system in the tropical Pacific having important consequences for weather around the globe. The measurements used in experiments have been gathered each ten minutes, from January 2006 to September 2006, by a moored buoy located in the Tropical Pacific.

We considered both a one and a three dimensional data stream. The *Rain* data set consists of 42,961 rain measurements. The *TAO* data set consists of 37,841 terns (SST, RH, Prec), where SST is the sea surface temperature, measured in units of degrees centigrade at a depth of 1 meter, RH is the relative humidity, measured in units of percent at a height of 3 meters above mean sea level, and Prec is the precipitation, measured in units of millimeters per hour at a height of 3.5 meters above mean sea level. The three attributes were normalized with respect to their standard deviation.

Finally, we employed the *1998 DARPA Intrusion Detection Evaluation Data* [13], that has been extensively used to evaluate intrusion detection algorithms. The data consists of network connection records of several intrusions simulated in a military network environment. The TCP connections have been elaborated to construct a data set of 23 numerical features. We used 50,000 TCP connection records from about one week of data.

In all experiments, the window size $W$ was set to 10,000 and the parameter $k$ was set to $0.005 \cdot W = 50$. The parameter $R$ was selected to achieve a few percent of outliers in the current window ($R = 0.1$ for *Gauss*, $R = 0.5$ for *Rain*, $R = 1$ for *TAO*, and $R = 1{,}000$ for *DARPA*).

Furthermore, an outlier query was submitted every one hundred objects. Measures reported in the sequel are averaged over the total number of queries submitted. The first query was submitted only after having observed the first $W$ data stream objects.

The classification accuracy of the method was evaluated. It is worth to recall that exact-STORM exactly detects distance-based outliers in the current window. Thus, the answer returned by this algorithm was used to evaluate the quality of the approximate solution returned by approx-STORM.

The *precision* and *recall* measures were employed. *Precision* represents the fraction of objects reported by the algorithm as outliers that are true outliers. *Recall* represents the fraction of true outliers correctly identified by the algorithm.

Figure 3 shows precision (dark bars, on the left) and recall (light bars, on the right) achieved by approx-STORM on the four considered data sets, for increasing values of $\rho$, that is $\rho = 0.01$, $\rho = 0.05$, and $\rho = 0.10$.

Interestingly, on the *Gauss* data set the method practi-

[1]See `http://www.pmel.noaa.gov/tao/`.

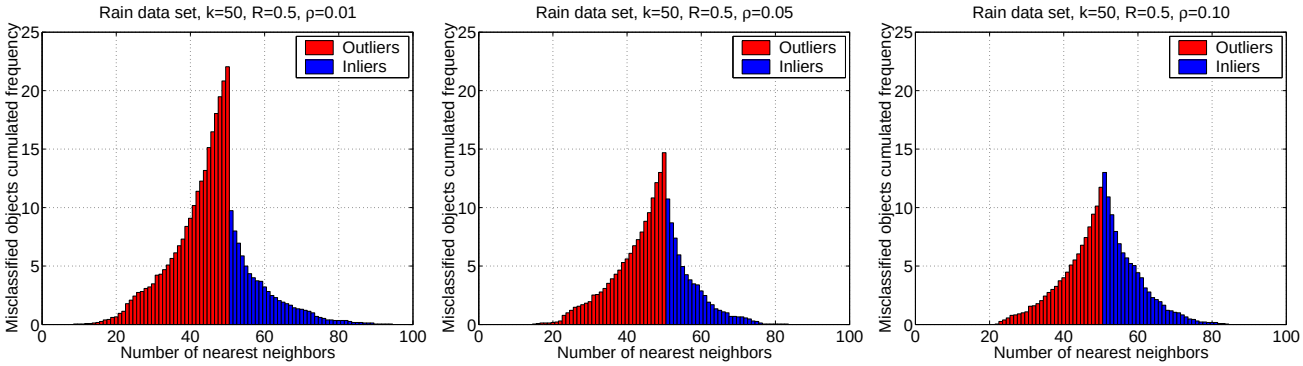



**Figure 4: Number of nearest neighbors associated with the misclassified objects of the *Rain* data set.**

| Data set | $\rho = 0.01$ | $\rho = 0.05$ | $\rho = 0.10$ | Exact |
|---|---|---|---|---|
| *Gauss* | 0.17 | 0.43 | 0.84 | 7.52 |
| *Rain* | 0.17 | 0.47 | 0.81 | 7.86 |
| *TAO* | 0.17 | 0.42 | 0.62 | 3.94 |
| *DARPA* | 0.17 | 0.29 | 0.47 | 3.28 |

**Table 1: Elaboration time per single object [msec].**

cally returned all and only the true outliers. This is because in this data set outliers are represented by noise which is well separated from the data distribution. Notice that other methods were not able to obtain this very good classification result.

As for the data sets from the TAO Project, since outliers there contained are associated to large oscillations of earth parameters, they lie on the boundary of the overall measurement distribution and are not completely separated from the rest of the population. Thus, there exists a region of transition where the approximate algorithm can fail to exactly recognize outliers (see below in this section for an evaluation of the characteristics of objects on which classification errors are made).

It is clear by the diagrams that by augmenting the parameter $\rho$ the precision tends to decrease while the recall tends to increase. This can be explained since by using a small sample size the number of nearest neighbors tends to be overestimated. Anyway, the classification accuracy was very good, e.g. precision 0.965 and recall 0.942 on the *Rain* data set, and precision 0.948 and recall 0.935 on the *TAO* data set, for $\rho = 0.05$.

The *DARPA* data set represents a challenging classification task due the considerable number of attributes it is composed of. The precision-recall trade-off previously observed is confirmed also on this data set. Moreover, the classification accuracy is of remarkable quality: for $\rho = 0.05$, precision 0.947 and recall 0.956 were achieved.

Table 1 reports the time (in milliseconds) employed by approx-STORM for various values of $\rho$ (first three columns), and by exact-STORM (fourth column) to process an incoming data stream object[2]. Approx-STORM guarantees time savings with respect to exact-STORM which are in most cases proportional to the parameter $\rho$. Differences in performances among the various experiments are justified by the different characteristics of the data sets, and among them, particularly, by the mean fraction $\eta$ of objects falling in the neighborhood of radius $R$ of data stream objects.

Figure 4 shows the distribution of the number of nearest neighbors associated with objects of the *Rain* data set which are misclassified by exact-STORM. These diagrams are interesting to comprehend the nature of the misclassified objects returned and the quality of the approximation. From left to right, diagrams are associated with increasing values of $\rho$.

The abscissa reports the number of nearest neighbors, while the ordinate the cumulated absolute frequency of misclassified objects. Two cumulated histograms are included in each diagram, one concerning outliers and the other concerning inliers.

Light bars (on the left) represent the mean number of outliers which are reported as inliers. Thus, these misclassifications concern the recall measure. Specifically, a bar of position $k_0$ and height $h_0$ represents the following information: among the objects having at most $k_0 (< 50)$ nearest neighbors (and hence outliers), on the average, $h_0$ of them have been recognized as inliers.

Dark bars (on the right) represent the mean number of inliers which are reported as outliers. Thus, these misclassifications concern the precision measure. Specifically, a bar of position $k_0$ and height $h_0$ represents the following information: among the objects having at least $k_0 (\geq 50)$ nearest neighbors (and hence inliers), on the average, $h_0$ of them have been recognized as outliers.

These diagrams show that for small sample sizes the number of errors is biased towards the outliers, due to the overestimation effect. Moreover, more interestingly, they show the nature of the misclassified objects. Indeed, as predicted by the analysis of Section 4.3, for an object the probability of being misclassified greatly decreases with the distance $|k_0 - k|$ between the true number $k_0$ of its nearest neighbors and the parameter $k$. Indeed, the majority of the misclassified inliers have a number of neighbors close to $k$. For example, when $\rho = 0.05$, almost all the misclassified outliers have at most 60 neighbors (compare this value with $k = 50$).

The quality of the approximate answer is thus very high. Although these objects are not outliers according to Definition 3.1, from the point of view of a surveillance application, they could be as interesting as true outliers, since they anyhow lie in a relatively sparse region of the feature space.

[2]Experiments were executed on a Core 2 Duo based machine having 2GB of main memory.

# 6. CONCLUSIONS

In this work the problem of detecting distance-based outliers in streams of data has been addressed. The novel *data stream outlier query* task was proposed and motivated, and both an exact and an approximate algorithm to solve it were presented. Also, bounds on the accuracy of the estimation accomplished by the approximated method. Finally, experiments conducted on both synthetic and real data sets showed that the proposed methods are efficient in terms processing time, and the approximate one is effective in terms of precision and recall of the solution.

**Acknowledgments.** The authors would like to thank the TAO Project Office for making available the collected measurements.

# 7. REFERENCES

[1] C. C. Aggarwal and P.S. Yu. Outlier detection for high dimensional data. In *Proc. Int. Conference on Managment of Data (SIGMOD'01)*, 2001.

[2] Charu C. Aggarwal. On abnormality detection in spuriously populated data streams. In *SIAM Data Mining*, 2005.

[3] F. Angiulli, S. Basta, and C. Pizzuti. Distance-based detection and prediction of outliers. *IEEE Transaction on Knowledge and Data Engineering*, 18(2):145–160, February 2006.

[4] F. Angiulli and C. Pizzuti. Fast outlier detection in large high-dimensional data sets. In *Proc. Int. Conf. on Principles of Data Mining and Knowledge Discovery (PKDD'02)*, pages 15–26, 2002.

[5] F. Angiulli and C. Pizzuti. Outlier mining in large high-dimensional data sets. *IEEE Transaction on Knowledge and Data Engineering*, 17(2):203–215, February 2005.

[6] A. Arning, C. Aggarwal, and P. Raghavan. A linear method for deviation detection in large databases. In *Proc. Int. Conf. on Knowledge Discovery and Data Mining (KDD'96)*, pages 164–169, 1996.

[7] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS*, pages 1–16, 2002.

[8] V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley & Sons, 1994.

[9] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proc. Int. Conf. on Knowledge Discovery and Data Mining (KDD'03)*, 2003.

[10] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proc. of the SIGMOD Conference*, pages 322–331, 1990.

[11] M. M. Breunig, H. Kriegel, R.T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *Proc. Int. Conf. on Managment of Data (SIGMOD'00)*, 2000.

[12] Edgar Chávez, Gonzalo Navarro, Ricardo A. Baeza-Yates, and José L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.

[13] Defense Advanced Research Projects Agency DARPA. Intrusion detection evaluation. In *http://www.ll.mit.edu/IST/ideval/index.html*.

[14] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. A geometric framework for unsupervised anomaly detection : Detecting intrusions in unlabeled data. In *Applications of Data Mining in Computer Security, Kluwer*, 2002.

[15] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Record*, 32(2):5–14, 2003.

[16] W. Jin, A.K.H. Tung, and J. Han. Mining top-n local outliers in large databases. In *Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'01)*, 2001.

[17] E. Knorr and R. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proc. Int. Conf. on Very Large Databases (VLDB98)*, pages 392–403, 1998.

[18] E. Knorr and R. Ng. Finding intensional knowledge of distance-based outliers. In *Proc. Int. Conf. on Very Large Databases (VLDB99)*, pages 211–222, 1999.

[19] E. Knorr, R. Ng, and V. Tucakov. Distance-based outlier: algorithms and applications. *VLDB Journal*, 8(3-4):237–253, 2000.

[20] Donald Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 1997.

[21] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proc. of the SIAM Int. Conf. on Data Mining*, 2003.

[22] S. Papadimitriou, H. Kitagawa, P.B. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *Proc. Int. Conf. on Data Enginnering (ICDE)*, pages 315–326, 2003.

[23] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B. Gibbons, and Christos Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *ICDE*, pages 315–326, 2003.

[24] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Proc. Int. Conf. on Managment of Data (SIGMOD'00)*, pages 427–438, 2000.

[25] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *International Conference on Very Large Data Bases*, Seoul, Korea, September 12-15 2006.

[26] O. Watanabe. Simple sampling techniques for discovery science. *TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems*, E83-D(1):19–26, 2000.

[27] Kenji Yamanishi, Jun ichi Takeuchi, Graham J. Williams, and Peter Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. In *KDD*, pages 320–324, 2000.