COPYRIGHT NOTICE

DOI: `http://dx.doi.org/10.1007/s10618-009-0159-9`.

Noname manuscript No.
(will be inserted by the editor)

Distance-based outlier queries in data streams: the novel task and algorithms*

Fabrizio Angiulli · Fabio Fassetti

Received: date / Accepted: date

Abstract This work proposes a method for detecting distance-based outliers in data streams under the sliding window model. The novel notion of one-time outlier query is introduced in order to detect anomalies in the current window at arbitrary points-in-time. Three algorithms are presented. The first algorithm exactly answers outlier queries, but has larger space requirements. The second algorithm is derived from the exact one, reduces memory requirements and returns an approximate answer based on estimations with a statistical guarantee. The third algorithm is a specialization of the approximate algorithm working with strictly fixed memory requirements. Accuracy properties and memory consumption of the algorithms have been theoretically assessed. Moreover experimental results confirmed the effectiveness of the proposed approach and the good quality of the solutions.

Keywords Data streams · Anomaly detection · Distance-based outliers

1 Introduction

In many emerging applications, such as fraud detection, network flow monitoring, telecommunications, data management and others, data arrive continuously, and it is either unnecessary or impractical to store all incoming objects.

* We state that a reduced version of the paper was published in the Proceedings of the ACM Sixteenth Conference on Information and Knowledge Management (CIKM 2007) under the title "Detecting Distance-Based Outliers in Streams of Data", but a substantial revision and extension was made and that the paper currently is not under review by another publication.

F. Angiulli · F. Fassetti
DEIS, Università della Calabria
Via P. Bucci, 41C
87036 Rende (CS), Italy
E-mail: {f.angiulli, f.fassetti}@deis.unical.it

In this context, an important challenge is to find the most exceptional objects among the incoming 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. As a matter of fact, stream monitoring applications are usually interested in analyzing the most recent behavior.

Therefore, data mining on 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* [17].

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 and extensively used 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 [10,8,19,12,18,1,25]. In particular, distance-based approaches [19] 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 [19,27,4] represent an useful tool for data analysis [20,15,23]. 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 [19]. This definition is slightly different from the original one provided in [19], which is: Given parameters p and R, an object is a distance-based outlier if at least fraction p of the objects in the input data set lies greater than distance R from the object. However, let n be the number of objects in the input data set, by setting $k = (1 - p) \cdot n$ the two definitions coincide.

Distance-based outliers have robust theoretical foundations, since they are a generalization of different statistical tests. Furthermore, they are compu-

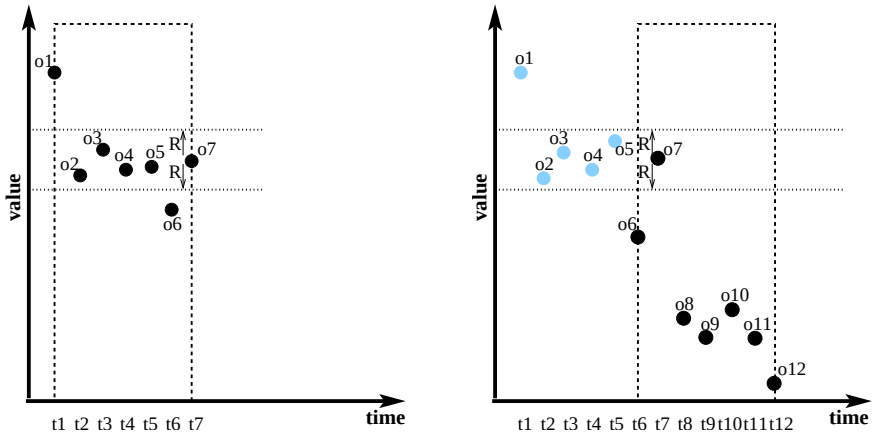
tationally efficient, since distance-based outlier scores are monotonic non-increasing functions of the portion of the data already explored.

The approach proposed in this work introduces a novel concept of *querying* for outliers. Specifically, previous work (see for example [31, 2, 28]) 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 the reader is referred to [9]).

The underlying intuition is that, due to evolution, stream characteristics 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, the *concept drift*, which is a typical and challenging characteristics of data 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 the concept drift can affect the outlierness of the data stream objects.

Example 1 Consider the following figures.



The two diagrams represent the evolution of a data stream of *one-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 diagram on the left reports the current window at time t_7 (comprehending the interval $[t_1, t_7]$), whereas the diagram on the right 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. Hence, if an analysis were required at time t_7, the object o_7 would be recognized as an inlier. Note that at time t_7 the object 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 in the current window since data stream objects assume lower values. In the figure on the right the evolution of the stream up to time t_{12} is reported. In the novel distribution, o_7 has no neighbors. Then, if an analysis were required at time t_{12}, o_7 should be recognized as an outlier. Note that at time t_{12} the object o_7 belongs to a very sparse region.

In this work, three algorithms for detecting distance-based outliers in data streams are presented. The first algorithm exactly answers outlier queries at any time, but has larger space requirements. The second algorithm is derived from the exact one, reduces memory requirements and returns an approximate answer based on estimations with a statistical guarantee. The third algorithm is a specialization of the approximate algorithm working with strictly fixed memory requirements.

The contribution of this work can be summarized as follows:

- the novel task of data stream outlier query is introduced;
- an exact algorithm, named STORM1, which efficiently detects distance-based outliers in the introduced model, is presented;
- an approximate algorithm, named STORM2, based on a trade off between spatial requirements and answer accuracy is derived from the exact one, which approximates object outlierness with a statistical guarantee;
- a fixed memory approximate algorithm, named STORM3, is introduced, which guarantees that the memory occupancy does not exceed an user provided buffer size;
- theoretical analysis of the technique is accomplished, assessing the accurateness of the introduced approximate methods, and their memory consumption;
- 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 approaches and algorithms for detecting outliers in data sets and data streams. Subsequent Section 3 formally states the data stream outlier query problem which this work deals with. Section 4 describes the three algorithms here introduced. Approximation properties and memory requirements are studied in Section 5. In Section 6 the implementation details of the algorithms are provided and both space and time costs are derived. Section 7 illustrates experimental results. Finally, Section 8 presents conclusions.

2 Related Work

Distance-based outliers have been first introduced by Knorr and Ng [19]. 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. As already noted above, this definition is a solid one, since it generalizes several statistical outlier tests. Some variants of the original definition and suitable mining algorithms have been subsequently introduced in literature. In particular, Ramaswamy et al. [27], 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 kth 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 kth nearest neighbor. In this work we will deal with the original definition provided in [19].

Several clever distance-based methods are designed to efficiently mine distance-based outliers [19,21,27,11,16,29,6,7], but they 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. Although most of the methods to detect anomalies in data mining consider the batch framework, some efforts have been made to address the problem of online outlier detection, and some of the proposed approaches are briefly surveyed next.

In [31] the SmartSifter system is presented. It addresses the problem from the viewpoint of the statistical learning theory. In particular, this system employs an online discounting learning algorithm to learn a probabilistic model representing the data source. Every time a datum is input, SmartSifter measures the outlierness of the datum by evaluating how much it is deviated with respect to a normal pattern. Specifically, the input datum o is defined as an outlier if the model learned after seeing o is significantly different from the model previously known.

In [2] the focus is on detecting anomalous events in a data stream in the presence of many spurious but similar patterns, where spurious events are rare in the stream but are likely to occur more frequently than the anomalous ones. Specifically, the proposed algorithm is a supervised one. Indeed, authors assume that an external source reporting actual anomalies is available. Then, when a rare event (which can be either anomalous or spurious) occurs, after a lag the external source reports it as anomalous or not. The introduced detection algorithm detects rare events on the basis of their deviation from expected values computed on historical trends, and tries to distinguish anomalous events from the spurious ones by exploiting the history of previous event occurrences. Clearly, such a prediction must be accomplished before the anomaly is notified by the external source. After the lag, the external source returns the right classification of the event, and, then, the algorithm exploits it to improve the accuracy of its future predictions. The algorithm must be initially fed with some historical data. At each event occurrence the algorithm performs two fundamental steps: first, it decides if the event is rare with respect to the historical trend; second, it classifies the event as either anomalous or spurious. Both checks are accomplished by using a statistical deviation based measure.

In [28] the problem of identifying outliers in streaming data acquired by a sensor network is addressed. In order to identify outliers, the distance-based and the MDEF [26] definitions are employed[1]. Authors adopt the sliding window model and propose a method to estimate the data distribution of sensor readings belonging to the most recent window. A hierarchically organization of the sensor network is assumed, where each parent node collects the measurements coming from its children. Authors have to tackle two difficulties: first, the sensor capabilities are limited, and, second, data coming from different sensors have to be combined together. These issues are dealt with by introducing a distributed framework to approximate data distributions. The estimation of the density distribution function f is performed by means of kernel density estimators and it is stored in the sensors. The distribution function f is exploited in order to identify an incoming object as an outlier. Specifically, when the distance-based outlier definition is employed, the number of objects (values) in the neighborhood of the observation at hand is evaluated through f, and, if this number is below an user defined threshold, then the observation is reported as an outlier. Conversely, when the MDEF-based definition is dealt with, the technique requires that each sensor stores, together with the estimation of the density distribution function f, a local estimation model for its input data stream. By means of these functions the deviation factor of the observation at hand can be evaluated.

It is worth to note that all the techniques above discussed, detect anomalies online as they arrive, and one-time queries are not supported. Moreover, in [2] an external source able to detect anomalies in the stream is required.

3 Statement of the Problem

In this section, the *Data Stream Outlier Query Problem* is formally stated. First of all, the definition of distance-based outlier is recalled [19].

Definition 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.

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.

[1] The Multi Granularity Deviation Factor (MDEF) is a measure of how the density in the neighborhood of an object compares with that of the objects in its neighborhood. An object is an outlier if the difference of its MDEF and the local average is statistically significant.

We assume that data stream objects are elements of a semimetric space, a generalized metric space on which is defined a distance function but the triangular inequality is not required to hold.

In the following we will use d to denote the space required to store an object, and Δ 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 denotes always the time of arrival of the last observed data stream object.

An *expired object* 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 2 (Data Stream Outlier Query) Given a data stream **DS**, a window size W, and parameters R and k, the *Data Stream Outlier Query* is: return the distance based outliers in the current window.

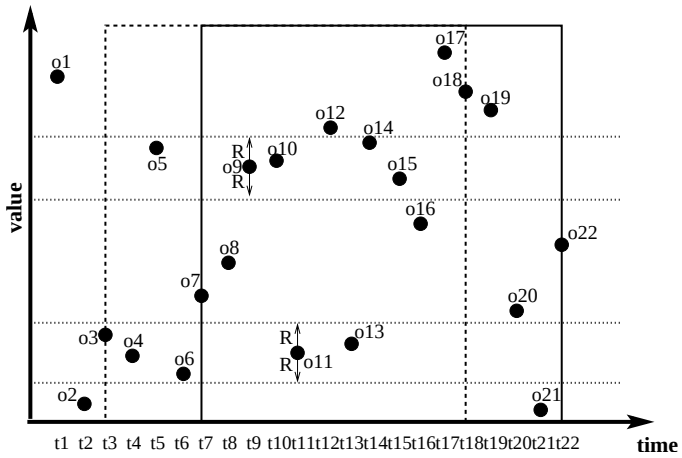
The time, at which a data stream outlier query is requested, is called *query time*.

In the following the neighbors of an object obj preceding it in the stream and belonging to the current window, are called *preceding neighbors* of obj, whereas the neighbors of an object obj following it in the stream and belonging to the current window are called *succeeding neighbors* of obj.

According to Definition 1, an *inlier* is an object obj having at least k neighbors in the current window. In other words, let α be the number of preceding neighbors of obj and β 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 at time t_{18} (the dashed one): 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}). But, since o_9 has three succeeding neighbors, it is a safe inlier, while o_{11} is not.

Indeed, consider for example instant t_{22} (the current window is the solid one). 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.

Three 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. The algorithm STORM1, working in this setting, is able to exactly answer outlier queries at any time (Section 4.1). However, in some applications, interesting windows can be so large that do not fit in memory, or in some other scenarios only limited memory can be allocated. In these cases, approximations must be employed. Algorithms STORM2 (Section 4.2) and STORM3 (Section 4.3) are designed to work in the latter setting by introducing effective approximations in STORM1.

4.1 Exact algorithm (STORM1)

The algorithm STORM1 is shown in Figure 1. It 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 outlier queries.

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. remove the oldest 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.

Fig. 1: The STORM1 distance-based outlier detection algorithm.

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

ISB provides a function *range query search*, that, given an object obj (also called *center*) and a real number $R \geq 0$ (also called *radius*), returns the nodes in ISB associated with objects whose distance from obj is not greater than R. We will detail the implementation of ISB in Section 6.

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 [22] 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, 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. Moreover, the list $n_{index}.nn\_before$ is deleted in order to reduce memory occupancy. 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.

As for the number *succ_neighs* of succeeding neighbors of $n.obj$, it 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 it is included in the answer of the outlier query.

4.2 Approximate algorithm (STORM2)

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.

Figure 2 shows the algorithm STORM2. In this algorithm two approximations are introduced with respect to STORM1.

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

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 $n_{index}.count\_after$;
   (b) if $n_{index}.count\_after \geq k$ (that is $n_{index}.obj$ is a safe inlier), 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;
5. if the number of safe inliers in ISB is greater than ρW, then remove from ISB randomly selected safe inliers till their number reduces to ρW.

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.

Fig. 2: The STORM2 distance-based outlier detection algorithm.

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 5.1, 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 ρ ($0 \leq \rho \leq 1$) 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 ρW, then a randomly selected safe inlier 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 for meaningful combinations of the parameters R and k, the number of outliers and of non-safe inliers, amounts to a negligible fraction of the overall population, as it will be both theoretically (in Section 5.3) and empirically (in Section 7) shown.

Thus, the number of nodes in ISB can be assumed approximately equal to ρW. In the following section it will be discussed how to compute an optimal value for ρ 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 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 α 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 α, 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, α 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 Approximate fixed-memory algorithm (STORM3)

It follows from the definition of Data Stream Outlier Query that in order to have the chance to return the correct solution to any future outlier query, all the objects which are currently outliers or non-safe inliers should be maintained into the ISB data structure.

The algorithm described in the preceding section reduces the memory by storing only a controlled fraction ρ of the safe inliers, with ρ depending on the approximation error estimation fixed by the user. As already noted above, for meaningful values of the parameters R and k the outliers and non-safe inliers amount to a negligible fraction of the overall population. In the next section, we will study the memory requirements of the above algorithm and provide conditions under which the memory used by it can be approximated to ρW. However, in some situations, that is when the above mentioned conditions do not hold, the number of non-safe inliers can be significant with respect to the value ρW.

Hence, in this section we introduce a fixed-memory approximation algorithm which takes care of the above scenario. The algorithm introduces a further approximation, concerning in estimating, among the non-safe inliers currently stored in the memory buffer, the objects most likely to become safe inliers.

The algorithm STORM3 builds on the algorithm STORM2 presented in the preceding section. It receives in input the additional parameter $\nu \in (0, 1]$, with $\nu > 2\rho$, representing the number of nodes that can be accommodated into the available memory buffer, that is νW. The memory buffer always maintains at most ρW nodes associated with safe inliers. The remaining part of the buffer, whose size is $(\nu - \rho)W$, maintains safe inliers, non-safe inliers, and outliers.

When the memory buffer is full and a new object arrives, an object of the buffer has to be released in order to free space for the incoming object. In this case, if the number of safe inliers exceeds ρW, then a randomly selected safe inlier is discarded in order to free space. Otherwise, that is if the number of safe inliers into the buffer is less than or equal to ρW, a priority is assigned with each non-safe inlier and outlier, in order to single out the object to be discarded.

This priority reflects the probability of the object of becoming a safe inlier. To this aim, the field $count\_after$ is exploited, and the probability that the object $n.obj$ will become a safe inlier is estimated as $\frac{n.count\_after}{t-n.id}$. In order to make this estimation robust, objects whose identifier is greater than $t - \rho W$ are not considered for deletion from the buffer. This strategy guarantees an highly accurate estimation of the likelihood of an object to become a safe inlier under the assumption that the population do not change, that is if the ρW succeeding neighbors represent a random sample of the future population. Indeed, in this case, the size of the sample employed for the estimation is at least the same than that employed for the estimation of preceding neighbors, and then the same statistical bound on the estimation error holds.

As for the algorithm pseudo code, STORM2 and STORM3 differ only for Step 5 of Figure 2. In particular, such step is modified in STORM3 in order to perform the procedure described in the following. When the memory buffer becomes full, if the number of safe inliers is greater than ρW, then a randomly selected safe inlier is removed from ISB. Otherwise, the priority $\pi(n) = \frac{n.count\_after}{t - n.id}$ is assigned to each node n stored in ISB associated with an outlier or a non-safe inlier $n.obj$ such that $n.id < t - \rho W$, and the node n^* such that $\pi(n^*)$ is maximum is removed from ISB.

5 Algorithm Properties

In this section the properties of the approximate algorithms are presented. In particular, first statistical bounds for the approximation error are stated (Section 5.1), then the misclassification probability law is derived (Section 5.2), and finally the memory requirements of STORM2 is analyzed by providing a way to estimate the number of safe inliers when the data stream distribution is known (Section 5.3).

5.1 Approximation error bounds

Next it is studied how to set the parameter ρ in order to obtain safe bounds on the approximation error estimation.

Let W be the window size, let w be the number of safe inliers in ISB, let α 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 μ denote the ratio $\frac{\alpha}{W}$.

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

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

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

This theorem asserts that, for any λ,

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

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

Consequently, if w is large enough, then the following relationship holds:

$$Pr\left[\frac{\widetilde{\alpha} - w\mu}{\sqrt{w\mu(1-\mu)}} \leq \lambda\right] \approx \Phi(\lambda). \tag{2}$$

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

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

$$w > \frac{\mu(1-\mu)}{\epsilon^2}\left(\Phi^{-1}\left(1-\frac{\delta}{2}\right)\right)^2 \tag{3}$$

then it satisfies (1).

Theorem 1 is a direct consequence of the central limit theorem (see [30] for details).

Proof Equation (2) is equivalent to:

$$Pr\left[\frac{\widetilde{\alpha}-w\mu}{\sqrt{w\mu(1-\mu)}} > \lambda\right] \approx 1-\Phi(\lambda) \tag{4}$$

and, since $\Phi(-\lambda) = 1-\Phi(\lambda)$, also to

$$Pr\left[\frac{\widetilde{\alpha}-w\mu}{\sqrt{w\mu(1-\mu)}} \leq -\lambda\right] \approx 1-\Phi(\lambda) \tag{5}$$

Starting from these relations and by setting

$$\lambda = \frac{\epsilon\sqrt{w}}{\sqrt{\mu(1-\mu)}}$$

it can be obtained that, for any $0 < \epsilon < 1$:

$$Pr\left[\frac{\widetilde{\alpha}}{w} > \mu+\epsilon\right] \approx 1-\Phi\left(\frac{\epsilon\sqrt{w}}{\sqrt{\mu(1-\mu)}}\right) \tag{6}$$

$$Pr\left[\frac{\widetilde{\alpha}}{w} < \mu-\epsilon\right] \approx 1-\Phi\left(\frac{\epsilon\sqrt{w}}{\sqrt{\mu(1-\mu)}}\right) \tag{7}$$

The goal (1) is equivalent to

$$Pr\left[\left|\frac{\widetilde{\alpha}}{w}-\mu\right| > \epsilon\right] < \delta$$

and, then, to

$$Pr\left[\frac{\widetilde{\alpha}}{w} > \mu+\epsilon\right] + Pr\left[\frac{\widetilde{\alpha}}{w} < \mu-\epsilon\right] < \delta$$

which, using relations (6) and (7), can be rewritten as

$$\left(1-\Phi\left(\frac{\epsilon\sqrt{w}}{\sqrt{\mu(1-\mu)}}\right)\right) + \left(1-\Phi\left(\frac{\epsilon\sqrt{w}}{\sqrt{\mu(1-\mu)}}\right)\right) < \delta$$

To conclude, from the above inequality, relation (3) is obtained.

Although the provided bound depends on the unknown value α, it can be safely applied by setting μ to $\frac{1}{2}$. Therefore, in order to satisfy (1), by substituting $w = \rho W$ in Formula (3), it follows that it is sufficient to set ρ to the value

$$\rho = \frac{1}{4\epsilon^2 W}\left(\Phi^{-1}\left(1-\frac{\delta}{2}\right)\right)^2. \tag{8}$$

It is worth to note that the bound for w given by expression (3) does not depend on the window size W. Furthermore, since in expression (8) the unknown value μ is safely set to $\frac{1}{2}$, whenever μ is different from $\frac{1}{2}$ the property (1) is guaranteed for values of ϵ and δ better than those used to compute w. In particular, the two following inequalities hold:

$$Pr\left[\left|\frac{\widetilde{\alpha}}{w}-\mu\right| \leq \epsilon\right] > 1-\delta^*, \text{ and} \tag{9}$$

$$Pr\left[\left|\frac{\widetilde{\alpha}}{w}-\mu\right| \leq \epsilon^*\right] > 1-\delta. \tag{10}$$

In the first inequality, δ^* 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, ϵ^* 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 μ is $\frac{1}{2}$, then $\delta^* = \delta$ and $\epsilon^* = \epsilon$.

It follows that, with probability $1-\delta$, the maximum error err that can made in computing α is

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

This value provides the maximum error made (with probability $1-\delta$) 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 the number of true preceding neighbors, when it arrives, is μW. As stream evolves, some preceding neighbors expire, and, assuming they are uniformly distributed along the window, their number becomes $\mu \cdot (W - t + id)$ in the portion of the stream preceding obj in the current window.

To correctly classify obj, if the sum between the number α of preceding neighbors of obj and the number β of its succeeding neighbors is greater (resp. smaller) than k in the current window, then also the estimated value for α plus the number β 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, let $\alpha = \mu\overline{W}$ be the true value of preceding neighbors of obj in the current window, let β 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:

$$\mu\overline{W} - 2\overline{W}\epsilon\sqrt{\mu(1-\mu)} + \beta > k. \tag{11}$$

Assuming that the distribution of succeeding neighbors is the same as the distribution of preceding neighbors, β can be approximated to $\mu(W - \overline{W})$, where $(W - \overline{W})$ is the portion of the stream in the current window following obj. Thus, by making μ explicit in Formula (11), it follows that for

$$\mu > \frac{kW + 2\overline{W}^2\epsilon^2 + \sqrt{(kW + 2\epsilon^2\overline{W}^2)^2 - k^2(W^2 + 4\overline{W}^2\epsilon^2)}}{W^2 + 4\overline{W}^2\epsilon^2} = \mu_{up} \tag{12}$$

an inlier is correctly 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

$$\mu < \frac{kW + 2\overline{W}^2\epsilon^2 - \sqrt{(kW + 2\epsilon^2\overline{W}^2)^2 - k^2(W^2 + 4\overline{W}^2\epsilon^2)}}{W^2 + 4\overline{W}^2\epsilon^2} = \mu_{down} \tag{13}$$

an outlier is correctly 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 the object obj is within 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 ϵ. Moreover, they depend on the current time t. As t increases, the error goes to zero and the interval 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.

5.2 Misclassification probability

In this section we will compute the probability that STORM2 misclassifies an inlier (Section 5.2.1) and an outlier (Section 5.2.2).

5.2.1 Inlier misclassification probability

We compute the probability of misclassifying an object which is an inlier. First, we consider the case in which the query is submitted when the object enters the current window. Suppose that the actual number of preceding neighbors μW of this object is greater than k. Then, we want that also the estimated number of preceding neighbors is greater than k. Therefore, the error $err = \mu W - \frac{\widetilde{\alpha}}{w} W$ should be such that $\mu W - err \geq k$.

Equation (2) can be rewritten as

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

and, then, as

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

Finally, we obtain

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

By setting

$$\gamma = -\lambda\frac{W\sqrt{\mu(1-\mu)}}{\sqrt{w}},$$

we get

$$Pr\left[err > \gamma\right] \approx \Phi\left(\frac{-\gamma\sqrt{w}}{W\sqrt{\mu(1-\mu)}}\right).$$

The probability of misclassifying an inlier is the probability that $err > \mu W - k$, hence, by substituting $\mu W - k$ to γ, we eventually get

$$Pr\left[err > \mu W - k\right] \approx \Phi\left(\frac{(k - \mu W)\sqrt{w}}{W\sqrt{\mu(1-\mu)}}\right). \tag{14}$$

Under the hypothesis that the stream is not currently changing its distribution, we can generalize formula (14) to obtain the misclassification probability for an inlier with a generic age.

In particular, the probability of misclassifying an inlier can be estimated by means of the law of total probability:

$$Pr\left[misclass\right] = \sum_{\eta} Pr\left[missclass \;\middle|\; age = \eta\right] Pr[age = \eta],$$

where the term $Pr[age = \eta]$ can be assumed equal to $1/W$, while the term $Pr\left[missclass \;\middle|\; age = \eta\right]$ can be derived by generalizing Equation (14), which

represents the case $age = 0$. Specifically, by following the same line of reasoning above shown, it can be obtained that for an object having $age = \eta$,

$$Pr\left[err > \gamma \mid age = \eta\right] \approx \Phi\left(\frac{-\gamma\sqrt{w}}{\overline{W}\sqrt{\mu(1-\mu)}}\right) \tag{15}$$

holds, where the error is defined as $err = \mu\overline{W} - \frac{\widetilde{\alpha}}{w}\overline{W}$.

In this case, the error should be such that $\mu\overline{W} - err \geq \overline{k}$, where $\overline{W} = W - \eta$ is the number of objects in the current window preceding the object having age η and $\overline{k}$ is the difference between k and the number of succeeding neighbors of the considered object. Then, the probability of misclassifying an inlier having age η is the probability that $err > \mu\overline{W} - \overline{k}$. The last inequality can be rewritten as:

$$err > \mu(W - \eta) - (k - \beta).$$

Since the number β of succeeding neighbors is $\mu\eta$, then the above inequality simplifies to $err > \mu W - k$, which is the same condition as above.

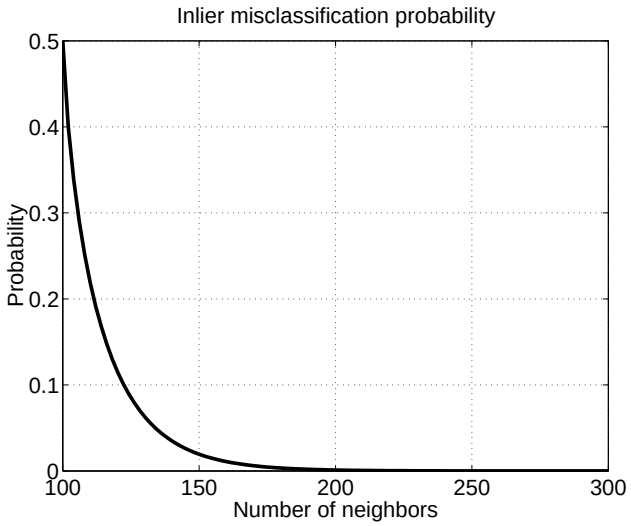
Hence, by substituting $\gamma = \mu W - k$ into Equation 15, we obtain

$$Pr\left[missclass \;\middle|\; age = \eta\right] = \Phi\left(\frac{(k - \mu W)\sqrt{w}}{(W - \eta)\sqrt{\mu(1-\mu)}}\right),$$

and, finally we get:

$$Pr\left[misclass\right] = \frac{1}{W}\sum_{\eta=1}^{W}\left[\Phi\left(\frac{(k - \mu W)\sqrt{w}}{(W - \eta)\sqrt{\mu(1-\mu)}}\right)\right]. \tag{16}$$

Example 3 For example, let the window size W = 10,000, let $\delta = 0.1$, let $\epsilon = 0.016$, and let $k = 100$. From Equation (3) we obtain that $w = 1088$ and, hence, that $\rho \approx 0.11$. The probability of misclassifying the inliers computed by means of Equation (16) is reported in the following figure:



It is clear from the above figure that the probability of misclassifying an inlier drastically decreases as the number of neighbors becomes greater than k.

5.2.2 Outlier misclassification probability

In this section the probability of misclassifying an object which is an outlier is computed. This probability can be obtained by following a line of reasoning very similar to that employed in the preceding section.

In this case, the error is defined as $err = \frac{\widetilde{\alpha}}{\widetilde{w}}\overline{W} - \mu\overline{W}$, and it should be such that $\mu\overline{W} + err < \overline{k}$, where $\overline{W} = W - \eta$ is the number of objects in the current window preceding the object having age η, and $\overline{k}$ is the difference between k and the number of succeeding neighbors.

Then, the probability of misclassifying an outlier aged η is the probability that $err \geq \overline{k} - \mu\overline{W}$. The last inequality can be rewritten as:

$$err \geq (k - \beta) - \mu(W - \eta),$$

and since the number of succeeding neighbors β is $\mu\eta$, then we get

$$err \geq k - \mu W.$$

The probability of misclassifying an outlier can be estimated exploiting the law of total probability:

$$Pr\left[misclass\right] = \sum_{\eta} Pr\left[missclass \;\middle|\; age = \eta\right] Pr[age = \eta],$$

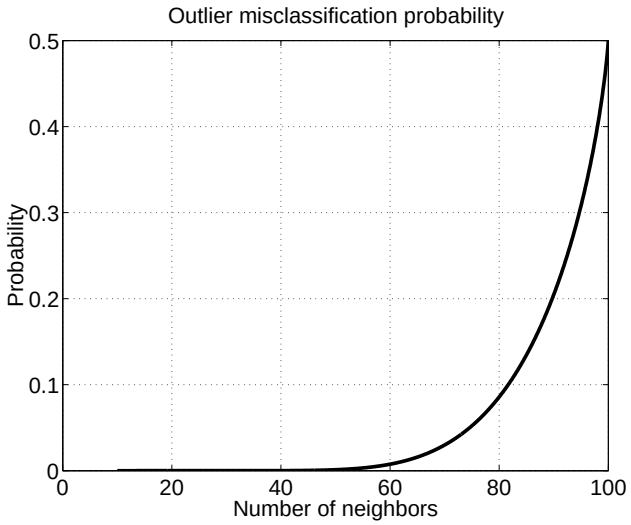
where the term $Pr[age = \eta]$ can be assumed equal to $1/W$, whereas the term $Pr\left[missclass \;\middle|\; age = \eta\right]$ is computable through an analysis similar to the previous case. Hence, we get that:

$$Pr\left[missclass \;\middle|\; age = \eta\right] = \Phi\left(\frac{(\mu W - k)\sqrt{w}}{(W - \eta)\sqrt{\mu(1-\mu)}}\right),$$

and, finally, that

$$Pr\left[misclass\right] = \frac{1}{W}\sum_{\eta=1}^{W}\left[\Phi\left(\frac{(\mu W - k)\sqrt{w}}{(W - \eta)\sqrt{\mu(1-\mu)}}\right)\right]. \tag{17}$$

Example 4 Consider the same setting as example 3. The probability of misclassifying the outliers computed by means of (17) is reported in the following figure:



Hence, the probability of misclassifying an outlier follows a trend similar to that of misclassifying an inlier.

5.3 Estimation of the number of safe inliers

In order to determine the memory requirements of STORM2, in this section we estimate the number of safe inliers falling into the current window.

The probability of being a safe inlier is the probability of having at least k succeeding neighbors. For an object *obj* we can think each incoming object as an independent Bernoulli trial, where the success is that the incoming object is a neighbor of *obj*, and the number of trials is the age of *obj*. Then, the probability for an object having age η of being a safe inlier can be modeled as the probability of having more than $k-1$ successes in η trials, or, equivalently, as the probability of not having less than k successes in η trials. Hence, the probability of being a safe inlier follows a binomial distribution:

$$Pr[o \text{ is safe} \mid \text{age} = \eta] = 1 - \sum_{i=0}^{k-1} p_{near}(o)^i (1 - p_{near}(o))^{\eta - i},$$

where p_{near} is the probability of a success, which corresponds to the probability that an incoming object is a neighbor of the object o.

The probability of being a safe inlier can, then, be estimated exploiting the law of total probability:

$$\begin{aligned} Pr[o \text{ is safe}] =& Pr[o \text{ is safe} \mid age = \eta] \cdot Pr[age = \eta] = \\ =& \frac{1}{W} \sum_{\eta=k}^{W-1} \left(1 - B_{k-1,\eta}(p_n(x))\right). \end{aligned} \tag{18}$$

where

$$B_{k,n}(p) = \sum_{i=0}^{k} p^i (1-p)^{n-i}$$

is the cumulative distribution function of the binomial distribution with parameters k (the number of successes) and n (the number of trials).

The probability p_{near} can be computed once the incoming data probability density function f is known. In particular, the probability $p_{near}(x)$ that an object x has a neighbor is

$$p_{near}(x) = F(x+R) - F(x-R),$$

where F denotes the cumulative distribution function associated with the data distribution f. Hence, the probability of being a safe inlier can be eventually obtained as

$$Pr[safe] = \int_{-\infty}^{+\infty} Pr[X = x] \cdot Pr[x \ is \ safe] \ \mathrm{d}x,$$

that is

$$Pr[safe] = \frac{1}{W} \int_{-\infty}^{+\infty} f(x) \sum_{\eta=k}^{W-1} (1 - B_{k-1,\eta}(F(x+R) - F(x-R))) \, \mathrm{d}x. \quad (19)$$

In the following we compute the value of the above probability for two common real data distributions, that are the Normal distribution and the Poisson distribution. Interestingly, it will be shown that the great majority of the objects in the current window are safe inliers. This result will be confirmed by the experimental results.

First, we need to recall the concept of unification between outlier definitions, introduced in [19] to formalize the concept that the distance-based outlier definition generalizes some statistical definitions for outlier. Specifically, the distance-based outlier definition *unifies* another definition *Def* for outlier, if there are specific values for parameters $k = \widehat{k} \cdot W$ ($\widehat{k} \in [0,1]$) and R such that an object is an outlier according to *Def* if and only if it is a distance-based outlier with parameters k and R.

5.3.1 Normal distribution

The normal distribution is one of the most important continuous probability distributions, since it models quantitative phenomena in many fields. The reason of its importance is mainly due to the central limit theorem which asserts that the normal distribution approximately occurs in many situations. For example, light intensity from a single source varies with time and it is usually assumed to be normally distributed, and also the thermal noise is approximately normally distributed. Moreover, normality is the central assumption of the mathematical theory of errors.

Consider a stream following a normal distribution with mean μ and standard deviation σ. In this case the probability density function is $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ and the cumulative distribution function is $\Phi(x)$ [24].

In [19], it is shown that distance-based outlier definition unifies the statistical outlier test for the normal distribution defined by [10] with parameters $\widehat{k} = 0.0012$ and $R = 0.13\sigma$.

As an example, consider a standard normal distribution. If W = 10,000, then $k = 12$ and $R = 0.13$. By substituting these values in (19), we get that $Pr[safe] = 0.9699$, that is the number of safe inliers is expected to be the 97.0% of the objects within the current window.

5.3.2 Poisson distribution

The popularity of the Poisson distribution is due to the fact that it models Poisson processes. Indeed, these processes apply to a large number of phenomena of discrete nature, which are well suited to be modeled with a stream. For example, the number of phone calls at a call center per minute, the number of times a web server is accessed per minute, or the emission of particles by a radioactive substance are all moldable as Poisson processes.

Consider a stream following a Poisson distribution with mean $\mu = 3$. In this case the probability density function is $f(x) = \frac{3^x e^{-3}}{x!}$ and the cumulative distribution function is $F(x) = \frac{\Gamma(\lfloor x+1 \rfloor, 3)}{\lfloor x \rfloor !}$ where Γ is the incomplete gamma function [24].

First we recall a well-known outlier test for Poisson distributions: let T be a set of objects Poisson distributed with parameter $\mu = 3.0$; an object x of T is an outlier if and only if $x > 8$ [10]. According to this test about the 3.8‰ objects are outliers. The distance-based outlier definition unifies this statistical test with parameters $\widehat{k} = 0.0027$ and $R = 0.5$.

Indeed, if x is an object of T, the expected percentage of neighbors within distance R from x is $Pr[x - 0.5 \geq X \geq x + 0.5]$, where X is a random variable distributed according to a Poisson distribution. Since the Poisson distribution is discrete, $Pr[x-0.5 \geq X \geq x+0.5]$ is equal to $Pr[X = x]$, that corresponds to the Poisson probability density function f evaluated in x. Hence, the expected percentage of neighbors for x is $f(x)$. From this equation, it can be obtained that for $x > 8$ the percentage of neighbors is expected to be lower than 0.0027 and then, that if x is greater than 8, x is expected to be an outlier.

As an example, if W = 10,000, then $k = 27$ and $R = 0.5$. By substituting these values in (19), we get that $Pr[safe] = 0.9719$, that is the number of safe inliers is expected to be the 97.2% of the objects within the current window.

6 Implementation Details and Cost Analysis

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

6.1 Implementation details

In the following, we assume that the stream objects belong to a certain metric space, but the algorithms work also with semi-metric spaces.

The ISB data structure is implemented as a *pivot-based* index [13]. 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 $\text{dist}(obj', obj) \leq R$ holds, 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 denoted as Δ. 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 $\mathcal{O}(\Delta \cdot \log N)$. As for the latter cost, let γ_{neigh} $(0 \leq \gamma_{neigh} \leq 1)$ 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 $\mathcal{O}(\Delta \gamma_{neigh} N)$. Supposing that the latter cost is always greater than the former one, the total cost for the range query search is $\mathcal{O}(\Delta \gamma_{neigh} N)$.

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 between 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.

6.2 Spatial analysis

For STORM1, 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, its identifier, and the counter of its succeeding neighbors. Recall that each object requires space d and each list requires space k.

For STORM2, ISB stores $(\rho + \zeta)W$ objects of the current window, where ζ denotes the fraction of non-safe inliers and outliers. Furthermore, for each stored object, the floating point number associated with its preceding neighbors (*fract_before*), its identifier, and the counter of its succeeding neighbors. Assuming meaningful combinations of the parameters, ISB stores approximately ρW objects, as confirmed by the analysis of Section 5.3.

Finally, for STORM3, ISB stores exactly νW objects of the current window, and, for each of them, the floating point number associated with its preceding neighbors (*fract_before*), its identifier, and the counter of its succeeding neighbors.

Summarizing, the spatial cost of the STORM algorithms is

- $\mathcal{O}(W(d + k))$ for STORM1,
- $\mathcal{O}(\rho W d)$ for STORM2, and
- $\mathcal{O}(\nu W d)$ for STORM3.

6.3 Temporal analysis

The procedure *Stream Manager* consists of different steps (see Figures 1 and 2). The cost of 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 $\mathcal{O}(\Delta\gamma_{neigh}N)$. For each of the $\gamma_{neigh}N$ objects returned by the search, STORM1 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, STORM2 and STORM3 perform only constant cost operations.

Step 4 consists in inserting a node into ISB and as previously noted has constant cost.

Finally, STORM2 and STORM3 include a fifth step, which consists in possibly removing a node from ISB. For STORM2, the removal cost is the cost of removing a safe inlier that consists in randomly selecting a safe inlier and in removing it from ISB, and both these operations have constant cost. Conversely, for STORM3, the removal cost is the cost of removing a safe inlier or the object most likely to become a safe inlier. In the former case, the removal cost is constant as for STORM2, whereas, in the latter case, the cost is linear to the size of ISB since, an ISB scan is required in order to compute the priority for the objects which are not safe inliers. Once the scan is accomplished, the cost for removing the object with highest priority from ISB is constant.

Summarizing, the cost of the procedure *Stream Manager* is

- $\mathcal{O}(\Delta\gamma_{neigh} W \log k)$ for STORM1,
- $\mathcal{O}(\Delta\gamma_{neigh} \rho W)$ for STORM2, and
- $\mathcal{O}((\Delta\gamma_{neigh} + 1)\nu W)$ for STORM3.

The procedure *Query Manager* consists of a scan of the ISB structure. For each node n, the STORM1 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, STORM2 and STORM3 perform only steps having constant cost. Summarizing, the cost of the procedure *Query Manager* is

- $\mathcal{O}(W \log k)$ for STORM1,
- $\mathcal{O}(\rho W)$ for STORM2,
- $\mathcal{O}(\nu W)$ for STORM3.

7 Experimental results

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

The rest of the section is organized as follows. Section 7.1 describes the employed data sets, subsequent Section 7.2 analyzes the accuracy of the algorithms in terms of both precision and recall and reports the memory requirements. In Section 7.3 the execution time of the algorithms is depicted. Section 7.4 provides a sensitivity analysis to the parameter k. The error distribution is discussed in Section 7.5. Finally, Section 7.6 studies the behavior of the algorithm when the concept drifts.

7.1 Datasets employed and experimental settings

Here, the data set employed are described, that are: *Mixed Gauss*, *Rain*, *TAO*, and *DARPA*.

The *Mixed Gauss* data set is a synthetically generated time sequence of 35,000 one dimensional observations, also used in [26]. 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)[2]. 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.

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

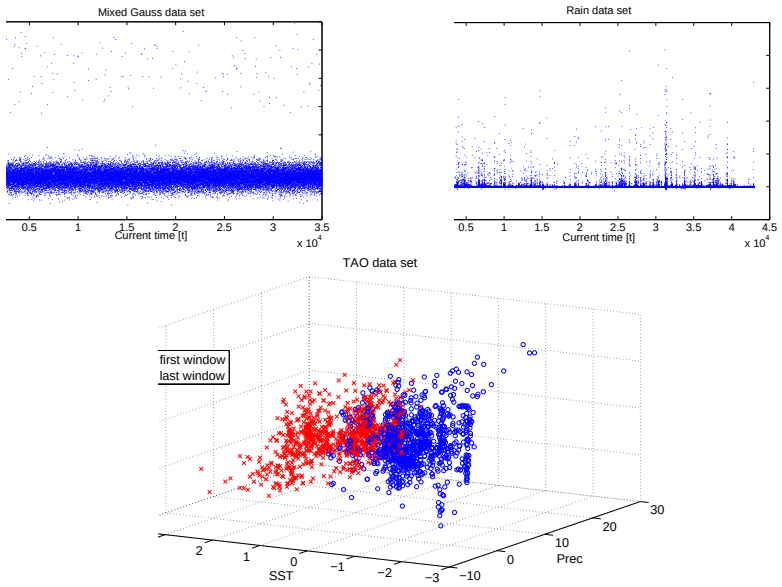


Fig. 3: Datasets employed.

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* [14], that has been extensively used to evaluate intrusion detection algorithms. Data consist 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.

The *Mixed Gauss*, *Rain*, and *TAO* data sets are shown in Figure 3. In particular, for the first two (one dimensional) data sets, we report the current time on the x-axis and the object value on the y-axis. Since *TAO* is a three dimensional data set, in order to visualize it, we show a 3D plot of the feature space reporting the objects in the first (x-marked) and in the last (o-marked) window.

Before leaving the section, we describe the experimental settings employed. In all experiments, if not explicitly specified, the window size W was set to 10,000 and the parameter k was set to $0.003 \cdot W = 30$, moreover, the parameter

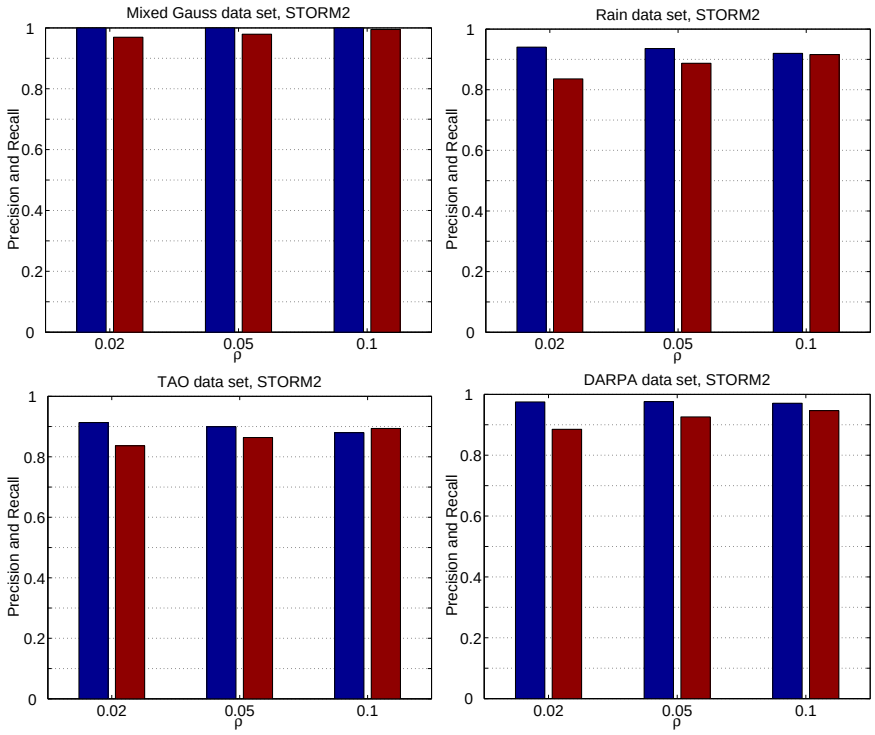


Fig. 4: Precision and Recall of STORM2.

R was selected to achieve a few percent of outliers in the current window ($R = 0.05$ for *Gauss*, $R = 5$ for *Rain*, $R = 2$ for *TAO*, and $R = 10{,}000$ for *DARPA*).

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

7.2 Classification accuracy and memory requirements

In this section the classification accuracy of STORM2 and STORM3 is evaluated. It is worth to recall that STORM1 exactly detects distance-based outliers in the current window. Thus, the answer returned by this algorithm was used to assess the quality of the approximate solution returned by STORM2 and STORM3.

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

ρ	Gauss	Rain	TAO	DARPA
0.02	0.0357 (+78.5%)	0.0359 (+79.5%)	0.0432 (+116.0%)	0.0384 (+92.0%)
0.05	0.0657 (+31.4%)	0.0659 (+31.8%)	0.0732 (+46.4%)	0.0684 (+36.8%)
0.10	0.1157 (+15.7%)	0.1159 (+15.9%)	0.1232 (+23.2%)	0.1184 (+18.4%)

Table 1: Index Max Size [%]

We start commenting on results of STORM2. Figure 4 shows precision (dark bars, on the left) and recall (light bars, on the right) achieved by STORM2, on the four considered data sets for increasing values of ρ, that are $\rho = 0.02$, $\rho = 0.05$, and $\rho = 0.10$.

Interestingly, on the *Gauss* data set for $\rho = 0.10$ the method practically returned all and only the true outliers, though the method behaves like the exact one also for smaller values of ρ. This is because in this data set outliers are represented by noise well separated from the data distribution.

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 ρ 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 0.934 and recall 0.887 on the *Rain* data set, and precision 0.899 and recall 0.864 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 tradeoff previously observed is confirmed also on this data set. Moreover, the classification accuracy is of remarkable quality: for $\rho = 0.05$, precision 0.976 and recall 0.926 were achieved.

Table 1 reports the actual memory occupancy of STORM2 on the previously mentioned data sets. For each data set, it is reported the occupied memory, as a fraction of the window, and, between parentheses, the memory actually required in addition to ρW, as a percentage of ρW.

As already noted in Section 4.2, the memory consumption of STORM2 depends both on the parameter ρ and on the number of non-safe inliers and outliers falling in the current window. From the table, it is clear that this number amounts to about the 2% of the window size for all the data sets. Thus, STORM2 required $0.02 \cdot W$ additional memory for all the executions. Clear enough, the lower the value of ρ is, the more this term influences the actual memory consumption of STORM2.

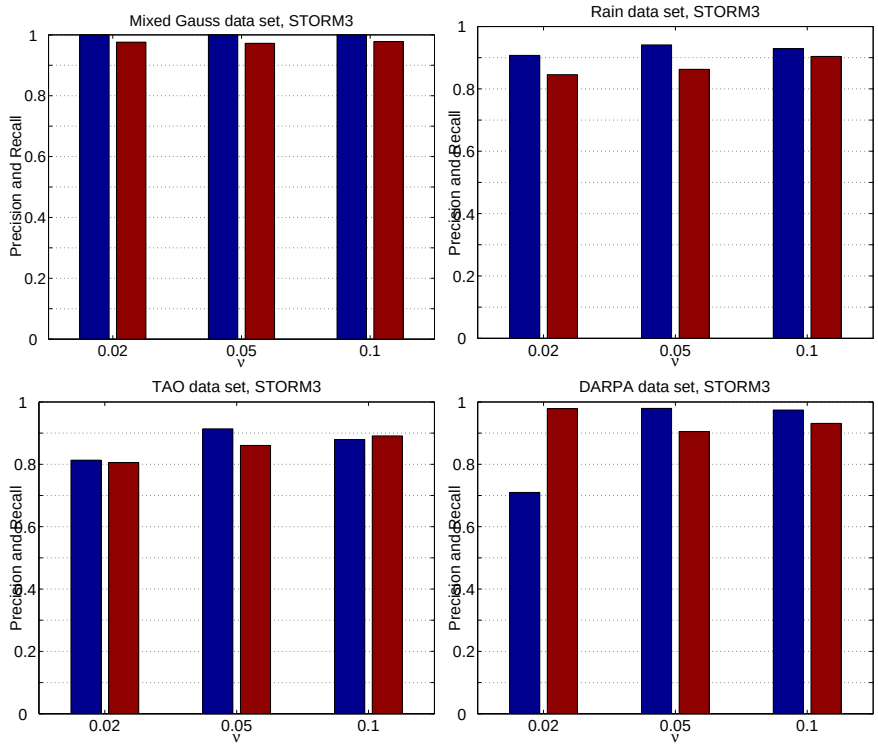


Fig. 5: Precision and Recall of STORM3.

Now, we comment on the results of the experiments using STORM3 (see Figure 5). In these experiments, we set ν to the same values above considered for ρ, that are $\nu = 0.02$, $\nu = 0.05$, and $\nu = 0.10$, while we set ρ to the value 0.3ν.

In order to understand the behavior of STORM3, it is important to analyze how the number of non-safe inliers and outliers influences the performances of STORM3. Consider the values of ν and ρ above defined. First, the number of safe inliers employed by STORM3 for the estimations is the 30% of the number of safe inliers employed by STORM2. Second, the number of non-safe inliers and outliers that can be maintained in memory for answering the queries is equal to $0.7\nu W$, and, from what above stated, the smaller the value of ρ (and, then, of ν), the greater the number of non-safe inliers and outliers.

For $\nu = 0.10$ and $\nu = 0.05$, the precision and the recall of STORM3 are very close to the values achieved by STORM2. Consider now the behavior of STORM3 for $\nu = 0.02$. From the above discussion, the performances of STORM3 are expected to degrade. In fact, the precision and the recall of STORM3 are smaller than those of STORM2. This is very evident for *TAO* and for *DARPA* which are high dimensional data sets, and thus the outliers in them are much more challenging to be detected. Conversely, for low dimen-

	$\rho, \nu = 0.02$		$\rho, \nu = 0.05$		$\rho, \nu = 0.10$		
Data set	STORM2	STORM3	STORM2	STORM3	STORM2	STORM3	STORM1
Gauss	0.17	0.12	0.25	0.21	0.47	0.41	4.74
Rain	0.19	0.15	0.35	0.30	0.71	0.59	8.96
TAO	0.20	0.14	0.36	0.26	0.69	0.57	7.34
DARPA	0.26	0.16	0.51	0.35	0.94	0.78	11.01

Table 2: Elaboration time per single object [msec].

sional data sets (*Gauss* and *Rain*) the loss of precision and recall appears to be negligible.

Summarizing, it can be concluded that, unless the parameter ρ is set to a very small value, STORM3 has performances comparable to those of STORM2, although it uses a fixed amount of memory, while STORM2 does not.

7.3 Execution Time

Table 2 reports the time (in milliseconds) employed by the presented algorithms to process an incoming data stream object[3]. Specifically, it is reported the time required by STORM2 for various values of ρ and by STORM3 for various values of ν, together with the time required by STORM1.

Both STORM2 and STORM3 guarantee time savings with respect to STORM1 which are in most cases proportional to the parameter ρ. 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 of objects falling in the neighborhood of radius R of data stream objects. Moreover, it can be noticed that STORM3 is slightly faster than STORM2, due to the fact that STORM3 works with strictly bounded memory, and then the number of objects stored in ISB by STORM2 is always greater than or equal to the number of objects stored by STORM3.

7.4 Sensitivity to parameter k

In this section we analyze the sensitivity of STORM to parameter k. To this aim we ran the STORM2 algorithm with three different values for this parameter, that are $k = 10$, $k = 25$, and $k = 50$. We set the value of the parameter R to have about 0.5% outliers in each window when $k = 10$ (the actual values of R are reported in Figure 6).

Figure 6 reports precision and recall on the previously introduced data sets. In general, for smaller values of k, the accuracy of the method slightly worsens. This can be explained by noticing that it becomes more sensitive to errors on the estimation of number of preceding neighbors (recall that in order

[3] Experiments were executed on a Pentium4 3.40 GHz machine having 2GB of main memory.

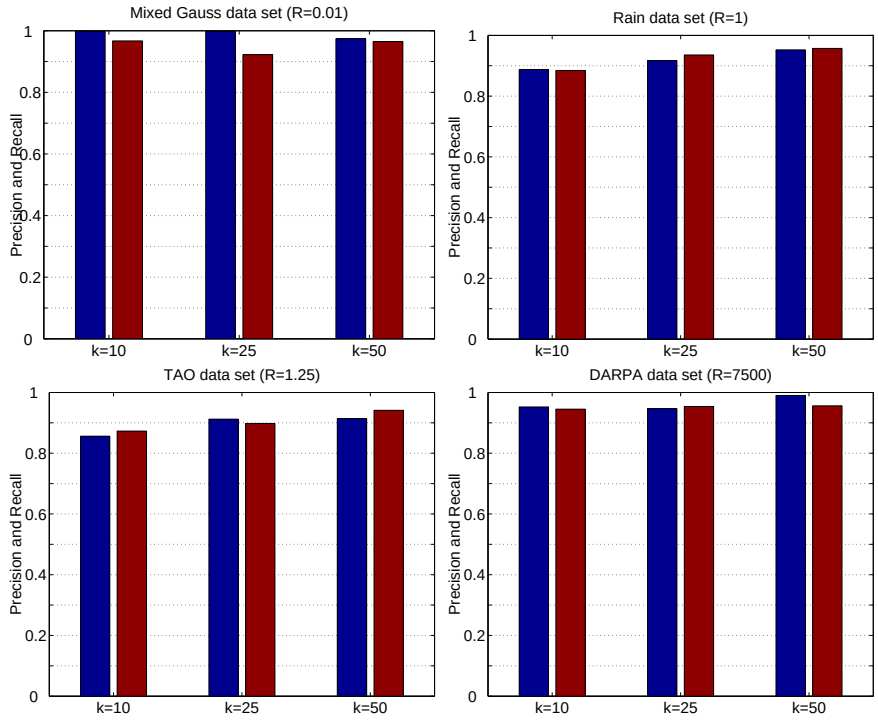


Fig. 6: Precision and Recall of STORM2 for various values of k.

to discriminate between inliers and outliers at query time the method has to decide if $prec\_neighs \geq k - succ\_neighs$ holds).

Moreover, since the value of the parameter R does not change, by augmenting the value of k the number of outliers increases. This has a positive effect on the precision and recall measures, since the effect of misclassified objects is mitigated by the greater number of outliers. The following table reports the average number of outliers in the windows:

Data set	$k = 10$	$k = 25$	$k = 50$
Mixed Gauss	60	78	124
Rain	64	129	211
TAO	49	105	160
DARPA	41	54	81

Before concluding, we note that although different values for the parameters k and R can be used, to obtain a meaningful analysis of the stream their values have to be set, either by the analyst or by an automatic parameter estimator, to proper combinations (e.g. as done for some standard distributions in Section 5.3).

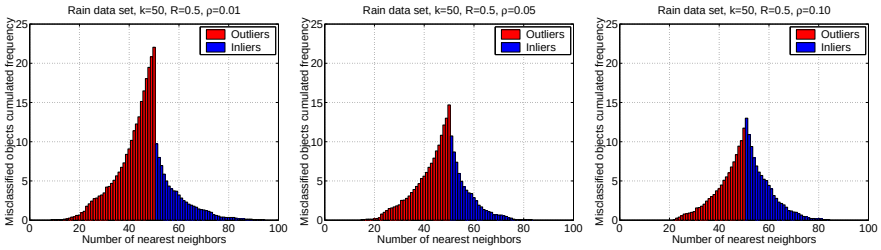


Fig. 7: Number of nearest neighbors associated with the misclassified objects of the *Rain* data set.

7.5 Approximation Error Distribution

Figure 7 shows the distribution of the number of nearest neighbors associated with objects of the *Rain* data set which are misclassified by STORM2. These diagrams are useful 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 ρ.

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

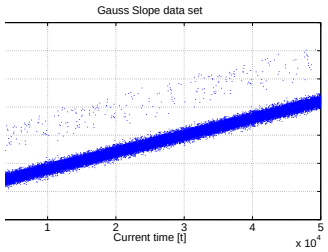
Light bars (on the left) represent the mean number of outliers which are reported as inliers. Thus, these misclassifications concern the recall measure. Specifically, a bar of position k_0 and height h_0 represents the following information: among the objects having at most $k_0 (< 50)$ nearest neighbors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

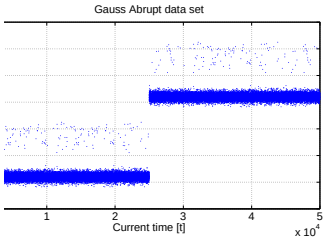
These diagrams show that for small sample sizes the number of errors is biased towards the outliers, due to the overestimation effect. Moreover, more interestingly, they show the nature of the misclassified objects. Indeed, as predicted by the analysis of Section 5.1, for an object the probability of 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 $k = 50$).

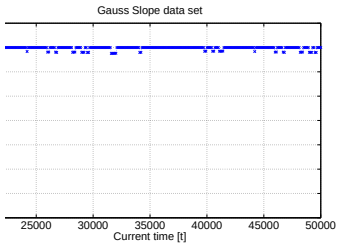
The quality of the approximate answer is thus very high. Although these objects are not outliers according to Definition 1, from the point of view of



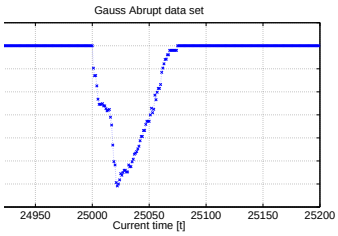
(a) *Gauss Slope* data set.



(b) *Gauss Abrupt* data set.



(c) *Gauss Slope*: F-score of STORM2.



(d) *Gauss Abrupt*: F-score of STORM2.

Fig. 8: Concept Drift Analysis

a surveillance application, they could be as interesting as true outliers, since they anyhow lie in a relatively sparse region of the feature space.

It is worth to notice that the analysis above accomplished agrees with the theoretical analysis concerning the misclassification probability depicted in Section 5.2.

7.6 Concept drift analysis

In this section we study the behaviour of the method when the concept drifts over time. With this aim we consider three data sets, that are the *Gauss Const* data set, the *Gauss Slope* data set, and the *Gauss Abrupt* data set, all composed of 50,000 points coming from a one dimensional gaussian distribution plus a cloud of outliers.

In the *Gauss Const* data set, the mean of the data distribution is held fixed over time to $\mu_0 = 1.1$. Conversely, the other two data sets present two different kinds of concept change. In the *Gauss Slope* data set, the mean of the data distribution slowly moves from μ_0 to $\mu_F = 2.6$ (see Figure 8a), while, in the *Gauss Abrupt* data set, the mean suddenly moves from μ_0 to μ_F at time $t = 25{,}000$ (see Figure 8b).

We ran the three STORM algorithms on the three data sets executing an outlier query at each instant of time. In all the experiments, we set k to 50 and R to 0.1. For STORM2 we set ρ to 0.1, while for STORM3 ν was set to 0.1 and ρ to 0.3. In order to visualize the accuracy of the method over time

we computed the *F-score* measure, defined as:

$$F\text{-}score = \frac{2 \cdot precison \cdot recall}{precision + recall},$$

which is a well-known combined measure of precision and recall.

On the *Gauss Const* data set, both STORM2 and STORM3 reported precision 1.0 and recall 1.0, and, hence, *F-score* 1.0.

Figures 8c and 8d report the *F-score* associated with STORM2 (the *F-score* associated with STORM3 is not reported since it almost all coincides with the F-score associated with STORM2) on the other two data sets.

As far as the *Gauss Slope* data set is concerned, the experiment shows that the method is practically insensitive to slow variations of the concept. Indeed, on this data set the *F-score* is almost always equal to 1.0 or close to 1.0 (see Figure 8c), which is the value obtained by the same algorithm on the *Gauss Fixed* data set.

The *Gauss Slope* data set represents a very difficult data drift scenario, in that at time instant $t = 25{,}000$ the overall data distribution moves to a different region of the space. Figure 8c shows the *F-score* of STORM2 for time t ranging from $t = 24{,}900$ to $t = 25{,}200$ (the *F-score* was 1.0 elsewhere). The figure shows that in correspondence of the concept change, the accuracy of the method deteriorates. In particular, the *F-score* worsens during the succeeding 22 time instants. Then, the method begins to learn the new distribution and the *F-score* improves till reaching value 1.0 at time instant $t = 25{,}075$. This experiment shows that, as expected, a period of time is needed to the method in order to adapt to sudden radical concept changes. However, interestingly, this transitory appears to be reasonably small (compare the length of the transitory, which is about 75, to the value of the parameter k, which is 50). Moreover, during this period the method performs reasonably well (the *F-score* is always above the value 0.7).

8 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 has been proposed and motivated, and both one exact and two approximate algorithms to solve it have been presented. Also, bounds on the accuracy of the estimation accomplished by the approximated method have been stated, and memory requirements for known distributions have been theoretically derived. Finally, experiments conducted on both synthetic and real data sets have shown that the proposed methods are efficient in terms processing time, and the approximate ones are 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.

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. Fabrizio Angiulli and Fabio Fassetti. Very efficient mining of distance-based outliers. In *CIKM*, pages 791–800, 2007.
7. Fabrizio Angiulli and Fabio Fassetti. Dolphin: An efficient algorithm for mining distance-based outliers in very large datasets. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(1):1–57, 2009.
8. 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.
9. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS*, pages 1–16, 2002.
10. V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley & Sons, 1994.
11. 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.
12. 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.
13. Edgar Chávez, Gonzalo Navarro, Ricardo A. Baeza-Yates, and José L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.
14. Defense Advanced Research Projects Agency DARPA. Intrusion detection evaluation. In *http://www.ll.mit.edu/IST/ideval/index.html*.
15. 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.
16. A. Ghoting, S. Parthasarathy, and M.E. Otey. Fast mining of distance-based outliers in high-dimensional datasets. In *Proc. of the SIAM International Conference on Data Mining (SDM'06)*, Bethesda, MD, USA, 2006.
17. Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Record*, 32(2):5–14, 2003.
18. 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.
19. 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.
20. 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.
21. E. Knorr, R. Ng, and V. Tucakov. Distance-based outlier: algorithms and applications. *VLDB Journal*, 8(3-4):237–253, 2000.
22. Donald Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 1997.
23. 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.
24. A.M. Mood, F.A. Graybill, and D.C. Boes. *Introduction to the theory of statistics*. McGraw-Hill, 1974.

25. 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.
26. 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.
27. 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.
28. 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.
29. Y. Tao, X. Xiao, and S. Zhou. Mining distance-based outliers from large databases in any metric space. In *Proc. of the Int. Conf. on Knowledge Discovery and Data Mining (KDD'06)*, pages 394–403, Philadelphia, PA, USA, 2006.
30. O. Watanabe. Simple sampling techniques for discovery science. *TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems*, 2000.
31. 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.