On Algorithms Selection for Unsupervised Anomaly Detection

Tommaso Zoppi, Andrea Ceccarelli, Andrea Bondavalli
Department of Mathematics and Informatics
University of Florence, Florence, Italy
{tommaso.zoppi, andrea.ceccarelli, bondavalli}@unifi.it

Abstract — **Anomaly detection, which aims at identifying unexpected trends and data patterns, has widely been used to build error detectors, failure predictors or intrusion detectors. Internal faults or malicious attacks have a different impact on the behavior of the system. They usually manifest as different observable deviations from the expected behavior, which may be identified by anomaly detection algorithms. Our study aims at investigating the suitability of unsupervised algorithms and their families in detecting either point, contextual or collective anomalies. To provide a complete picture, we consider both sliding and non-sliding window algorithms which operate in unsupervised mode. Along with qualitative analyses of each algorithm and family, we conduct an experimental campaign in which we run each algorithm on three state-of-the-art datasets in which we inject either point, contextual or collective anomalies. Results show that non-sliding algorithms are capable to detect point and collective anomalies, while they cannot effectively deal with contextual ones. Instead, sliding window algorithms require shorter periods of training and naturally build a local context, which allow them to effectively deal with contextual anomalies. Such observations are summarized to support the choice of the correct algorithm depending on the investigated class(es) of anomaly.**

Keywords — anomaly detection, unsupervised learning, sliding window, algorithms, data mining

I. Introduction

Modern systems such as cyber-physical infrastructures, Systems-of-Systems or Cloud environments are composed of several software layers and a multitude of services. Notoriously, these complex systems have to deal with component failures or misbehaviours, or may be targeted by cyber-attacks, requiring attentive security countermeasures.

Consequently, error detectors [42], [20], intrusion detectors [5], [12] and failure predictors [8] were proposed to enhance system dependability and security by analysing system data. They aim at identifying error-prone, malicious or unauthorized activities assuming that a fault, or an attack, has distinguishable effects on such data, generating observable deviations from the expected behaviour. Detectors may apply signature-based techniques [5], which consist of checking properties or looking for predefined patterns (*signatures*) in monitored data to detect the manifestation of a fault, or an ongoing attack. Signature-based approaches have good detection capabilities when dealing with known faults [42] or attacks [2], [5], but they may fail in identifying unknown faults. Moreover, when an unknown fault or a *zero-day attack* [1] (i.e., an attack that exploit novel or undiscovered system vulnerabilities) is revealed, a new signature must be defined and added to the signatures set.

To deal with unknown faults or zero-day attacks, research moved to techniques suited to detect unseen, novel attacks. Anomaly detectors are intended to *find patterns in data that do not conform to the expected behaviour of a system* [2]: such patterns are known as *anomalies*. Once an expected behaviour is defined, anomaly detectors aim at identifying deviations from such expectations, providing a mean to discover known attacks, zero-day attacks [12] and emerging threats [27]. However, anomaly detection is particularly effective only when the expected behaviour can be defined precisely. Along with an appropriate data quality, selecting the correct detection algorithm(s) represents a key decision when defining an anomaly detector.

More precisely, each algorithm has specific strengths and weaknesses related to its ability in identifying anomalies. As in [2], data points may be unexpected at all (*point anomaly*), unexpected in a given scenario (*contextual anomaly*), or anomalous as a group (*collective anomaly*). Typically, algorithms building a global notion of expected behaviour are effective in identifying point anomalies (global outliers), while algorithms building a local expected behaviour effectively deal with contextual (local outliers) [2] and, often, collective anomalies (groups of data points that are anomalous as a whole) [3], [4]. Further, although most of the algorithms have a generic, system-independent definition, they are more effective on specific systems, since the same fault may manifest slightly different anomalies from system to system. Amongst all the possible algorithms, in this study we focus on *unsupervised anomaly detection algorithms*, which do not require labels in training data [40], [15]. Unsupervised algorithms aim at partitioning the dataset in two classes: the expected data points and the anomalous ones. They consider anomalies are rare events, and therefore the majority of the data points in the dataset should describe an expected behaviour. We refer to 6 families of unsupervised algorithms, namely: *clustering*, *statistical*, *classification*, *neighbour-based*, *density-based* and *angle-based*.

It is worth remarking that most of the available algorithms require massive amounts of training data, along with high computational power and, consequently, long training periods. However, when systems evolve or update their internal parameters, the expected behavior changes. This calls for a brand new training phase of detection algorithms, which might be excessively time-consuming. To cope with these issues, there is a noticeable amount of research that was carried on in the recent years [41], [42] regarding *sliding window algorithms*, or rather algorithms that rely on light training phases, allowing them to promptly react to changes or updates of the system, adapting their parameters to suit the current behavior. Sliding windows are particularly useful to capture the local expected behavior, and to limit the processing time of the algorithms, which build their knowledge starting from a limited set of system observations, or data points.

Our work aims at *investigating how well different algorithm cope with specific classes of anomalies*. We i) first investigate suitability of algorithms to classes of anomalies by reviewing the literature of unsupervised anomaly detection algorithms, and ii) then conduct quantitative analyses to confirm or deny our initial analyses. We select a total of 12 unsupervised algorithms through literature review, a non-sliding and a sliding window algorithm for each of the 6 families [2], [10]. The selection criteria favour well-known, consolidated algorithms with public implementations rather than novel proposals.

More in detail, we qualitatively analyse each algorithm to extract information related to their ability in detecting either *point*, *contextual* or *collective* anomalies. Then, we conduct an experimental campaign to confirm or deny the conjectures built at the first stage. Results show that non-sliding algorithms effectively detect point and collective anomalies. Moreover, some algorithms are strongly influenced by the choice of input parameters, and therefore may provide either very good or very bad results depending on such parameters. Sliding window algorithms have worse overall detection capabilities, but require shorter periods of training and naturally build a local context, which ultimately allow them to effectively detect contextual anomalies.

The remaining of this paper is structured as follows: Section II presents basics and a review of works related to anomalies, anomaly detection and sliding window strategies. Section III describes the selected unsupervised algorithms, whose suitability in detecting anomalies is qualitatively discussed in Section IV. Section V presents the methodology and the results of our experimental campaign. Qualitative and quantitative results are discussed in Section VI, while Section VII concludes the paper, elaborating on possible future works.

II. Basics and Related Works

A. On the Nature of Anomalies

In the paper we will refer to the observation of the state of the system at a given instant as a *data point.* Each data point is composed by *f* feature values, which are collected for all the *f* observed features. Feature values are used to determine if the data point is anomalous. More in detail, anomalies are *rare data points* that may be classified as [2]:

- *point* anomaly (outlier): a single data point that is out of scope or not compliant with the trend of a variable, e.g., enormous amount of calls to a (web)service;
- *contextual* anomaly: a data point that is unexpected in a specific context, e.g., snow during summer in Italy;
- *collective* anomaly: a collection of related data points that is anomalous with respect to the entire trend or dataset e.g., rhythm breakings in heartbeats.

More in detail, point anomalies, or *global outliers*, can be detected by algorithms that derive patterns [6] or statistical-based methods that reconstruct the statistical inertia of the data under investigation [7]. Instead, contextual anomalies, or *local outliers*, require knowledge on the current state of the system, because they identify data points that are unexpected in a given context. Known algorithms gather historical [7], user/operator-provided [4], or runtime [22] data to define a “contextual” expected behavior that is used to evaluate monitored data. Finally, collective anomalies may be hard to detect when extensive training sets are not available [2]: they represent a group of data points that are neither point anomalies nor contextual anomalies, but that are unexpected as a whole e.g., rhythm breakings in heartbeats. Therefore, these anomalies can be detected only by algorithms that are able to consider individual observations as correlated data points [3], [4].

B. Unsupervised Anomaly Detection

Different anomaly detectors may be instantiated depending on the nature of the data of a target system [2]. If fully-labeled training data is available, *supervised anomaly detection* may be adopted [43]. Labelled data points are used to train an algorithm using both expected and anomalous data points that have already been reported. When labelled anomalous data points are unavailable, incomplete or not trustable, it is still possible to construct an expected behavior in a *semi-supervised* fashion. Lastly, when training data is not labeled at all or not available, the only option is an *unsupervised anomaly detection* approach [10], [40], [44]. As remarked in [2], semi-supervised techniques can be adapted to operate in an unsupervised mode by using a sample of the unlabeled data set as training data, as in [15]. Such adaptation *assumes that the test data contains very few anomalies* and the training process is robust enough.

Noticeably, when configuring an anomaly detector for a target system, we can assume that a fully labeled training set will not be available in most of the cases due to i) lack of trustable labeling techniques, ii) difficulties in gathering reliable data, or iii) dynamic and evolving characteristics of the system and its workload, which calls for adaptive data analysis solutions. Overall, unsupervised algorithms have a wider application range than supervised and semi-supervised algorithms, despite having lower detection capabilities due to the absence of labels in training data.

C. Sliding Windows

In specific contexts, we can define anomaly detectors that use *sliding windows*, a sequence of *n* elements automatically updated as time passes [41], [23]. Sliding window may be i) *count-based*, containing the *n* most recent data points, or ii) *time-based*, containing the data points within a fixed time interval covering the most recent timestamps. Such sliding windows are particularly useful to capture the local expected behavior, and to limit the processing time of the algorithms, which analyzes a maximum of *n* elements. This way, the scope is limited to an interval over the entire set of collected data, and, periodically, novel data points are appended to the window, while older points are discarded as they become less relevant to the analysis.

Window Update Policy. Policies to manage the window have been investigated in the literature. If the window is managed as a *First In, First Out* (FIFO) queue, it may contain anomalies, negatively affecting the definition of the local expected behavior, and decreasing the overall ability in detecting anomalies. To cope with this problem, the window may be updated only if specific conditions are met [24], [23] e.g., the value to be added in the window is not an anomaly. This improves the quality of the data in the window, but requires additional computation at each step.

Window Size. Broadly, a small window reduces detection time and detection capabilities, while a large window increases detection time and detection capabilities. Small windows contain only a few recent data points and are usually efficient in characterizing the current context. Point anomalies need more knowledge on the system, because they are deviation from the global behavior of the system, which can hardly be described by a restricted number of data points in the window. An optimal setup for detecting collective anomalies is not always easy to identify. Overall, the more knowledge we have, the more accurate is the characterization of the expected behavior, leading to higher possibilities in detecting all classes of anomalies.

III. Selection of Algorithms

This section reports on the unsupervised anomaly detection algorithms we select for our comparative study. We survey the state of the art looking for i) families of unsupervised anomaly detection algorithms, ii) a non-sliding window algorithm for each family, and iii) a sliding window algorithm for each family.

A. The Selection Criteria

Our selection criteria favor well-known, consolidated algorithms with public implementations. These criteria are defined considering that our study is oriented to a fair comparison of families of algorithms rather than the comparison of recent and advanced strategies. Despite technical advancements that may have been proposed by domain experts, we assume that algorithms belonging to a given family have their intrinsic strengths and weaknesses that may be mitigated, but that cannot be removed at all. Comparing consolidated versions of algorithms allow us to evaluate how the baseline idea behind the algorithms of a given family suits the detection of specific classes of anomalies. Then, when improved algorithms are proposed, we expect results related to the suitability of basic ones to anomalies to maintain their validity.

B. Families of Algorithms

After a literature review [2], [12], we identify six families of algorithms, briefly described below. The selected algorithms are summarized in Table I and Table II, along with their strengths and weaknesses. Details are provided in Section III-C to Section III-H.

- *Neighbor-based* algorithms learn by analogy: they label a data point as anomalous or expected depending on the label of its nearest neighbor(s), considering an *f*-dimensional space. The way the label is chosen may be the same as a nearest neighbor, or the majority over the *k* nearest neighbors.
- *Clustering* algorithms partition a set of data points in such a way that data points in the same group (cluster) share similar characteristics. Data points that cannot be assigned to any of the existing clusters, or that do not met specific inclusion criteria, are anomalous.
- *Angle-based* algorithms relate data to high-dimensional spaces, and measure the variance in the angles between the data point to the other points [9]. This is a technique with good scalability. Expected data points have a large angle variance, while anomalies typically result in very small variance in the angles from couples of points.
- *Classification* algorithms identify the class a new data point belongs to, depending on information collected during previous activities e.g., assigning a given email into spam or non-spam classes. Despite they are mainly used in supervised setups, there are also some unsupervised approaches [15], [25].
- *Density-based* algorithms estimate the density of the neighborhood of each data point. When a data point differs from the expected ones, it lies in a low-density area and it is then labeled as anomalous.
- *Statistical algorithms are* based on the assumption that only expected data points occur in high probability regions of a given statistical distribution. These techniques fit a distribution to the expected points, and then apply statistical inference to determine if a novel data point belongs to this distribution or not. In unsupervised mode, statistical techniques do not generally assume knowledge of underlying distribution, which is derived as data is computed.

C. Neighbor-based Algorithms

Non-Sliding Algorithm - ODIN: stemming from kNN [13], this distance-based method was designed to identify point anomalies. For each data point, kNN examines the whole dataset to determine their feature distances to the given point. This allows isolating *k* nearest neighbors (NN), creating the so-called *kNN graph*. The *Outlier Detection using Indegree Number* (ODIN, [11]) algorithm improves kNN by defining as anomalies the data points that have a low number of in-adjacent edges in the kNN graph.

Sliding Window Algorithm - SNN: SNN [23] expands two techniques, namely i) conceptual partitioning for nearest neighbor monitoring over update streams, and ii) the skyline maintenance in the distance-time space, partially pre-computing future evolutions of the nearest neighbors. The k-skylines are matched with the kNN. Together, these two techniques allow the continuous kNN monitoring over both time-based and count-based sliding windows.

D. Clustering Algorithms

Non-Sliding Algorithm – KMeans: K-means [17] assigns data points to *k* subsets, or *clusters*, by their feature values. First, *k* centroids are randomly initialized and each data point is assigned to the cluster with the nearest centroid. Centroids may be updated, fitting evolving scenarios also in unsupervised mode. Finally, data points that are too far from the centroid of their cluster are labeled as anomalies.

Sliding Window Algorithm - SWClustering: this sliding window clustering [18] extends the classic clustering algorithms by tracking the evolution of clusters. *SWClustering* combines the exponential histogram with the temporal cluster features to handle the in-cluster evolution, i) keeping a high quality of the clusters, ii) requiring less memory and low overheads, and iii) eliminating the influence of outdated data points.

E. Angle-based Algorithms

Non-Sliding Algorithm - FastABOD: detects anomalous data points depending on the angles between pairs of distance vectors to other points [9]. For each data point, the algorithm first calculates the *Angle Based Outlier Factor* (ABOF) to its k-nearest neighbor as the normalized scalar product of the difference vectors of any triple of neighbors. According to [9], the usage of kNN provides a better approximation. Then, *FastABOD* ranks the data points according to their ABOF. The smaller the ABOF, the bigger the probability that the data point represents an anomaly.

Sliding Window Algorithm - ABSAD: To mitigate the impact of anomaly-irrelevant attributes, in *angle-based subspace anomaly detection* (ABSAD, [22]) the ABOF is computed in a subspace that captures the most information about the discordance of an object to its adjacent ones. The subspace is matched by a sliding window. As data flows, the window is updated aiming to maintain a meaningful subspace. For each data point, the deviation from its neighborhood is evaluated as ABOF on the subspace.

F. Classification Algorithms

Non-Sliding Algorithm – One Class SVM: this algorithm conducts semi-supervised anomaly detection [2] aiming to learn a decision boundary [14]. However, *One-Class SVMs* can be used for unsupervised anomaly detection: a support vector machine is trained with the dataset and then each data point is classified considering the normalized distance of the data point from the determined decision boundary [15].

Sliding Window Algorithm – iForestASD: iForestASD [24] is an adapted version of *iForest* [25] with linear time complexity and low memory requirement, and it does not use any distance or density measure. Since anomalies are rare events with feature values that differ from expected data points, they may be isolated closer to the root of the tree instead of the leaves, which is typical of expected data points. Moreover, even if each constituent *Isolation Trees* cannot detect all the anomalies, their ensemble *Isolation Forest* can balance such weaknesses.

G. Density-based Algorithms

Non-Sliding Algorithm – LOF: *Local Outlier Factor* (LOF) [16] compute the kNN for each data point, and use them to calculate the density index, called *Local Reachability Density* (LRD). The anomaly score is then obtained by comparing the LRD of a data point with the LRD of its kNN. Expected data points have scores close to 1.0, while anomalies usually result in bigger scores.

Sliding Window Algorithm - INCLOF: LOF assign each data point an anomaly score. *INCLOF* [19], an incremental variant of LOF algorithm, updates this score whenever the sliding window is updated. Additionally, the quality of the data in the window is maintained by i) omitting anomalies

TABLE I. NON-SLIDING ALGORITHMS

Algorithm	Family	Strengths	Weaknesses
ODIN [11]	Neighbour	Tracks a picture of k-neighbourhoods, which helps in detecting point anomalies.	If an outlier is too close to some neighbouring inlier, it can be misclassified.
K-Means [17]	Clustering	*k* Clusters are calculated as data flows, good in identifying single anomalies.	Converges to a local optimum: good for contextual anomalies. This negatively impacts on the detection of point anomalies
FastABOD [9]	Angle	Robust to high dimensionality data points, tailored to identify point anomalies	If the amount *k* of neighbours is too low, Anomalies other than point may not be detected.
One-Class SVM [15]	Classification	A well-defined boundary allows detecting all anomalies with good overall capabilities.	Outliers in the training set negatively impact the definition of the boundary, which may not be devised correctly.
LOF [16]	Density	Good overall capabilities. Local density score is normalized and easy to analyse.	Strongly depends on the size *k* of the neighbourhood.
HBOS [7]	Statistical	Histograms help in identifying point anomalies.	Hard to characterize the current context, and collective anomalies may be difficult to detect.

TABLE II. SLIDING WINDOW ALGORITHMS

Algorithm	Family	Strengths	Weaknesses
SNN [23]	Neighbour	Continuous kNN monitoring, especially good for point anomalies.	With small k values, groups of anomalies are likely to be ignored
SWClustering [18]	Clustering	Effectively tracks evolving clusters to identify contextual and, often, collective anomalies.	Collective anomalies are likely to be considered as novel small clusters.
ABSAD [22]	Angle	Effectively extracts multi-point subspaces (contexts)	It becomes asymptotically stable when using many *f* features.
iForestASD [24]	Classification	Each constituent isolation tree is built for identifying (isolated) point anomalies.	Data points should not break the “isolation” assumption. iForest has several input parameters.
INCLOF [19]	Density	Adaptation of LOF to sliding windows. Good Overall Capabilities.	Need an accurate management of data in the window to keep detecting anomalies.
SPS [20]	Statistical	Detects individual data points that do not follow to the expected behaviour.	High variance of data increases the safety margin, exposing to missed detections.

from the window, and ii) optimizing the update of LOF scores. This activity is particularly useful to release memory resources and to avoid polluting expected data.

H. Statistical

Non-Sliding Algorithm - HBOS: This approach [7] generates an histogram for each feature by using the values of all the available data points. The anomaly score is computed by multiplying the inverse heights of the columns in which each feature of the data point reside. Such technique assumes that the investigated features are independent, making HBOS fast even when dealing with large datasets [10]. If features are dependent, such dependencies need to be neglected.

Sliding Window Algorithm - SPS: the *Statistical Predictor and Safety Margin* (SPS, [20]) algorithm predicts an interval, given by a minimum and a maximum value, in which the next value is expected to fall. The interval is built by combining i) the last observed feature value, ii) a *prediction* of its evolution, and iii) a *safety margin*, which grows when feature values show high variance. SPS identifies values that do not follow the statistical inertia of the observations, and consequently it is suitable for detecting point anomalies [21].

IV. Suitability of Algorithms to Classes of Anomalies

To the best of our knowledge, there is no detection algorithm that can always effectively identify anomalies. To such extent, we discuss here how the design choices characterizing each family of algorithm impact their ability in identifying specific classes of anomalies. Results are built as an analytical process, which is complemented by a quantitative evaluation described in Section V.

Each family of unsupervised anomaly detection algorithms has their own peculiarities. More in detail, algorithms belonging to *neighbor-based*, *angle-based* and *density-based* families are primarily intended to identify point anomalies, while they may not be able to detect collective anomalies if the size k of the selected neighborhood is smaller than the size of the collective group of anomalies. Indeed, an indegree score may help detecting them, as demonstrated in [11]. This issue is shared also with *clustering* algorithms, since a collective anomaly may lead them to create a separate cluster for the particular group of data points. However, some clustering algorithms [18] mitigate this problem by labeling as anomalous both data points i) belonging to small clusters and ii) far from known clusters. Instead, one of their main weaknesses is on detection of contextual anomalies: a data point with feature values that occurred frequently in the past but that should not occur anymore is likely to fall into a cluster of expected data points.

When considering sliding windows, the usage of a reduced portion of past data limits the overall capabilities of characterizing a global expected behavior. As a consequence, the overall ability of algorithms in detecting point anomalies may decrease since they may not be able to characterize a global notion of expected behavior when using small windows. Moreover, algorithms belonging to *neighbor-based*, *angle-based*, and *clustering* families are more likely to identify contextual anomalies, since they build their knowledge on a limited portion of recent data which, in fact, represent the current context.

In unsupervised anomaly detection, *classification-based* algorithms are often identified as *One-Class SVM* [15]. This algorithm learns a non-linear boundary that is then used to partition expected data points and anomalous ones. However, if the data used to create such boundary does not follow the assumptions (e.g., no outliers in the training data, *Gaussian* kernel), its detection capabilities will decrease dramatically. An alternative classification is provided by *Isolation Forests* [25], i.e., ensembles of *Isolation Trees*, which are also adequate for sliding windows [24]. Despite it has a lower but comparable performance with SVMs, it is more robust to noise in training data. Lastly, *statistical* algorithms are based on probability distributions and statistical analyses: it is therefore difficult to identify a general capability in detecting classes of anomalies.

V. Experimental Evaluation

Here we describe our process, the main steps, and the results of our experimental study of anomaly detection algorithms targeting specific classes of anomalies.

A. Methodology

Considering the unsupervised algorithms and algorithm families that are summarized in Section III, we create some datasets for scoring the performances of the algorithms when dealing with specific classes of anomalies. To the authors' knowledge, there is no work that provides effective and robust links between safety or security threats and the classes of anomalies they usually generate. Therefore, existing datasets that log intrusions or the manifestation of internal faults of a system are not fitting our needs. To fix this issue, we

i. take expected data coming from 3 state-of-the art datasets [29], [30], [26] that are commonly used for anomaly-based intrusion detection,
ii. remove non-numerical features, if any, and
iii. inject data perturbations which simulate faults or attacks manifesting as either point, contextual, or collective anomalies.

This generates 3 *sub-datasets* for each of the initial ones. A sub-dataset is extracted from the initial dataset with the artificial injection of either point, contextual or collective anomalies. Sub-datasets are used to train the parameters and then to run the selected algorithms, ultimately evaluating the capabilities of each algorithm in detecting each class of anomaly. Multiple instances of the investigated algorithm are generated during the training phase. Each algorithm instance is assigned to a different dataset feature, or set of features. One or more instances may be run in parallel, combining their results according to a given strategy. Such process is expanded in Section V.D.

Lastly, we present results by means of metrics such as *Precision*, *Recall*, *False Positive Rate*, *Accuracy*, *F-Score*, *Matthews Coefficient* and *Area Under ROC Curve*.

B. Datasets

The data we use in this paper is based on the publicly available datasets NSL-KDD [29], ISCX [30] and UNSW-NB15 [26], which were published in the last 10 years. These datasets report on network traffic logged on different systems and with different purposes. We filter such datasets obtaining subsets of respectively 57592, 122148, and 44353 data points. We preprocess each dataset to characterize a general notion of expected behavior for each feature e.g., *minimum* and *maximum* values, *average*, *median*, *variance*, and we modify such data by inserting additional records which contain anomalous values for some features of a data point. Despite such artificial anomalies do not perfectly replicate real anomalies [2], we simulated them to the best of our capabilities as follows:

- *Point anomalies*: some feature values are updated with values that are either smaller than the minimum values or bigger than the maximum values logged in the dataset for a given feature.
- *Contextual anomalies*: some feature values are updated with values that are not contained in the 95% confidence interval, considering the last 100 feature values as current context.
- *Collective anomalies*: in our experiments a collective anomaly is a set of three subsequent data points where we updated some feature values with values outside the 80% confidence interval, considering the last 100 feature values as current context. Collective anomalies are not subsequent point or contextual anomalies; therefore we used a different confidence interval to generate them.

We inject anomalies in the sub-datasets, obtaining a 95% - 5% ratio of expected – anomalous data points. The resulting sub-datasets are freely available at [35].

C. Metrics

The effectiveness of anomaly detectors is usually analyzed using indicators counting correct detections - *true positives* (TP), *true negatives* (TN) - and the wrong ones, either missed detections (*false negatives*, FN) or wrong detections (*false positives*, FP). These indicators are commonly used to derive the so-called *confusion matrix*. Aggregate metrics based on the abovementioned indicators are *Precision*, *Recall* (or *Coverage*), *False Positive Rate*, *Accuracy*, *F-Score*(β) [31] and *Matthews Coefficient (MCC,* [32]). Especially in the F-Score(β), varying the parameter β makes possible to weight the precision with respect to the recall (note that F-Score(1), or *F1*, is referred as *F-Measure*), allowing to give more relevance to either FPs or FNs. It is worth remarking that when dealing with critical systems, we may prefer reducing the occurrence of missed detections (FN), even at the cost of a higher rate of FP. Since we want our study to be as generic as possible, we investigate Accuracy, F-Measure and MCC, which aggregate TPs, TNs, FPs, and FNs considering FPs and FNs with the same weight. However, it is acknowledged [39] that, under specific circumstances, metrics as F1 or Accuracy "*can be misleading, since they do not fully consider the size of the four classes of the confusion matrix in their final score computation*". Therefore, as suggested in [39], in our study we choose MCC as reference metric.

In addition, we calculate the *Area Under ROC Curve (AUC)*, which is used [33], [34] to estimate how the choice of the parameters of a given algorithm impacts its detection capabilities. Depending on the system, it may be difficult to find an optimal combination of input parameters. Therefore, the choice of input parameters may lead to a high variance of detection scores and, consequently, lower AUC scores.

D. Instances Generation and Anomaly Threshold

As described in the methodology, we generate multiple instances of each algorithm during training. Instances are assigned to the numeric features of the dataset. An algorithm instance can be assigned to i) a single feature, ii) a subset of features, or iii) the *n-dimensional* space composed by all the available features. In our study, we create subsets of features when two or more features show a *Pearson Correlation* [38] above 90%, either positive or negative, always including the set composed by all the available features.

The usage of multiple instances assigned to different features helps improving the algorithm performances since it allows choosing the instance which show better scores during training as the one that will be used in the validation phase. Moreover, more than one instance can be run in parallel at validation phase, collecting their anomaly evaluations separately, and then voting the single scores, according to a given *anomaly threshold*. During training, different thresholds are considered, namely

- ALL: an anomaly alert is raised if and only if all the algorithm instances label the current data point as anomalous;
- HALF: half of the algorithm instances must evaluate the feature values assigned to them as anomalous;
- ONE: an anomaly alert is raised if at least one algorithm instance detects anomalous feature values.

It is important to remark that the choice of this threshold heavily affects the overall detection performance. Considering a single instance (i.e., adopting ONE threshold) usually reduces the amount of false negatives; instead, the need of a consensus among instances, e.g., ALL threshold, reduces the number of (false) alarms.

E. Experimental Setup

First, we retrieve available public implementations of the selected algorithms. *KMeans*, *ODIN*, *LOF*, *FastABOD* and *OneClass SVM* are extracted from the ELKI [36], while the *HBOS* implementation is devised starting from the paper [7]. Regarding sliding windows algorithms, no public implementations were made available by authors, except for SPS. Therefore, since sliding windows algorithms are usually adapted versions of known algorithms with optimized execution time, we simulate the sliding window algorithms as follows. Instead of considering the experimental data all-at-once, we simulate a sliding window of different *sizes* WS = {10, 20, 50, 100}, and every time a new data point is added to the window, we run the corresponding algorithm by using the content of the window as training set. Also, we consider as *policies* to slide the windows i) FIFO (see Section II.C), and ii) FIFONormal, which extends the FIFO policy by sliding the window only if the current data point is not an anomaly. We employ as underlying algorithms: *ABOD* for simulating *ABSAD*, *LOF* for *INCLOF*, *kNN* for *SNN*, *KMeans* for *SWClustering*, and *IsolationForest* (i.e., WEKA [37]) for *iForestASD*.

The experiments are conducted according to the methodology presented in Section V.A. Parameters tuning is employed during training to find an optimal setup of each instance of a given algorithm. Tuning is performed by i) first, running different combinations of parameters; ii) then, comparing results for the different parameters. For example, we run *kNN* and kNN-dependent algorithms i.e., *ODIN*, *FastABOD*, with k ϵ {1, 2, 3, 5, 10, 20, 50, 100}. In the followings we will also elaborate on the relevance of the choice of parameters through discussion of AUC scores.

Datasets with artificial injection of anomalies are created according to the process described in Section V.B. Starting from the initial portions of NSL-KDD, ISCX, UNSW datasets, non-numeric features were removed, leaving 38 features for NSL-KDD, 6 for ISCX, and 39 for UNSW. During training phase, we respectively create 38, 20, and 25 feature subsets according to *Pearson Correlation*. This results in 77 algorithm instances for NSL-KDD, 27 for ISCX, and 65 for UNSW, which sum i) the single features, ii) the feature subsets, and iii) the set composed by all the available features. All the instances are evaluated together during training phase and all the possible algorithm configurations are ranked according to a given target metric, in our case *MCC* (see Section V.C). The ranked list is then used in the evaluation phase to maximize algorithm scores.

The data generated during the execution of the algorithms was initially stored in CSV files, and successively analyzed with *MS Excel* for fast analytics, such as comparing metric scores. The experimental campaign is executed on an Intel Core i7, 32GB of RAM server, with at least 15GB and up to 20GB dedicated to the execution of the experimental campaign. Overall, computing all the scores and metrics required approximately three weeks of 24h execution. A complete view of the collected data cannot be presented here for brevity: the files we used to aggregate metric data and plot graphs are available at [35].

TABLE III. AVERAGE METRIC SCORES FOR NON-SLIDING ALGORITHMS

Non-Sliding Algorithm	# Conf	FPR	P	R	F1	ACC	*MCC*	AUC
OneClass SVM [14]	12	0.21%	0.97	0.88	0.92	0.99	*0.92*	0.65
LOF [16]	8	1.49%	0.84	0.89	0.86	0.98	*0.85*	0.67
ODIN [11]	8	2.45%	0.77	0.87	0.81	0.97	*0.80*	0.72
HBOS [7]	9	1.60%	0.84	0.77	0.80	0.97	*0.78*	0.68
KMeans [17]	8	4.41%	0.74	0.84	0.78	0.95	*0.74*	0.71
FastABOD [9]	16	2.64%	0.62	0.63	0.62	0.94	*0.58*	0.81

F. Results: Non-Sliding Algorithms

Table III show the metric scores we obtain by running the selected non-sliding algorithms to our three synthetic datasets. The table aggregates scores by algorithm, ranking the resulting rows according to a decreasing MCC.

Figure 1 reports on the MCC scores for non-sliding algorithms, for each class of anomalies. We observe that SVM shows the highest scores. However, we can observe how algorithms that have low metric scores in Table III are really effective in detecting specific classes of anomalies, i.e., *KMeans* and *FastABOD*. Point anomalies are detected well by *FastABOD* (see column with squares in Figure 1), while *KMeans* show very good capabilities in identifying collective anomalies (see horizontal-striped columns in Figure 1). These two algorithms also show high AUC scores, meaning that the impact of the choice of the parameters used for training phase does not heavily affect performances. Instead, we observe that the AUC for *One-Class SVM* (see Table III), is the lowest out of the selected non-sliding algorithms.

LOF and, to a lesser extent, *ODIN*, are good generalist algorithms, which do not show specific weaknesses for identifying each class of anomaly. Lastly, we remark how *HBOS*, despite not optimal in terms of metric scores, is a light and fast algorithm, making it useful when a massive amount of computational resources is unavailable.

G. Results: Sliding Window Algorithms

Figure 2 depicts the MCC scores of the 6 selected sliding window algorithms, grouped by class of anomalies. Observing the vertical-striped columns in the figure it is possible to realize that such algorithms do not really suit the detection of point anomalies, despite the neighbor-based algorithm *SNN* achieves the highest average MCC score. Instead, contextual anomalies are detected fairly well by

Fig. 1. Average MCC scores for non-sliding algorithms, for each class of anomalies. Error bars represent standard deviation among the datasets.

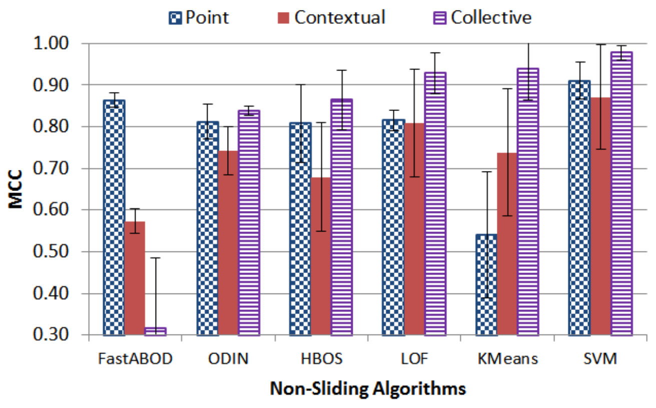


Fig. 2. Average MCC scores for sliding window algorithms, for each class of anomalies. Error bars represent standard deviation among the datasets.

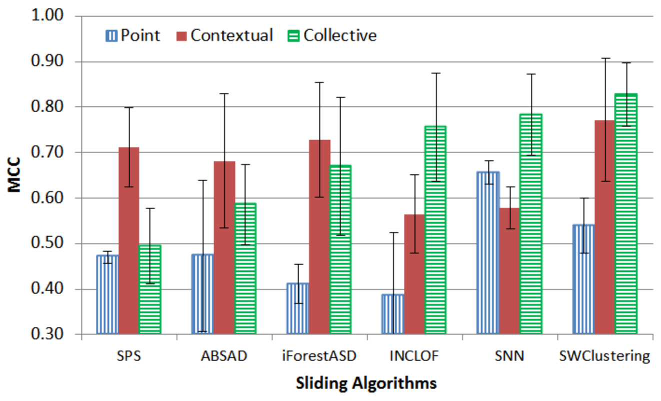
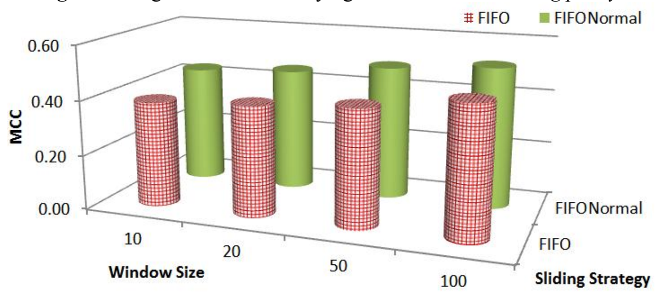


Fig. 3. Average MCC scores varying *window size* and *sliding policy*.



SPS, *iForestASD*, *SWClustering* algorithms; the latter is the sliding window algorithm with the higher average MCC score with collective anomalies (see horizontal-striped columns in Figure 2).

To complete our analysis on sliding window algorithms, in Figure 3 we report the average MCC when varying the window size and the sliding policy. We can notice how the MCC score increases as the window size, or rather the amount of system knowledge, grows. In addition, the *FIFONormal* policy prevents anomalous data points to enter the window, allow defining a more precise expected behavior. This approximately results in 10% higher average MCC scores with respect to the former *FIFO* policy.

VI. Analysis of the Results and Findings

We analyze here the results reported in the previous section, summarizing the main findings in Section VI-D.

A. *Analysis of the Results*

At a first glance, results in Figure 1 do not match the expectations. We expect point anomalies to be the easiest to detect, and collective anomalies to be the hardest to detect. Instead, their average MCC score is almost the same, while contextual anomalies are detected with more difficulties. We explain this result as follows: the selected algorithms use large training sets, which allow a careful and precise definition of the boundaries between anomalous and expected behavior. However, during their process they derive a global boundary, which does not fit the detection of local anomalies, or rather data points that have feature values that are not expected in a given context. As a result, contextual anomalies cannot be detected easily. An in-depth view of metric scores related to each non-sliding algorithm is reported in Table III. Starting from the top of the table, we can observe how *One-Class SVM* turns out to have better overall metric scores. However, depending on the data points used for training, SVMs may not be able to correctly derive a boundary, nullifying its detection ability. Therefore, it is important to look for other options – a different algorithm or a combination of two or more that may be run simultaneously - that provide similar metric scores, and based on more robust training processes.

Overall, we can observe how sliding window algorithms have lower scores than non-sliding ones. In fact, the usage of a sliding window limits the amount of training data. While it allows building faster algorithms, such restricted knowledge of the system negatively impacts the detection capabilities. Comparing the scores in Figure 1 and Figure 2, we can notice that sliding window algorithms are not really effective in detecting point anomalies, which need a global notion of expected behavior to be identified. Instead, they identify contextual anomalies with scores comparable with non-sliding algorithms. More in detail, they generate more false positives i.e., higher FPR and lower Precision, but similar Recall scores. Sliding window algorithms show fairly good capabilities in detecting collective anomalies, although adopting non-sliding algorithms is preferable i.e., average MCC score of 0.81 with respect to 0.69 of sliding window algorithms.

As a side remark, we do not report on the impact of anomaly thresholds, which instead are logged in our data at [35]. In fact, while considering different thresholds helps improving metric scores, it is not possible to extract a rule to derive an optimal threshold depending on our data. Consequently, we do not expand this discussion since it may be confusing, as it is not substantiated by detailed and extensive sensitivity analyses (planned as future work).

B. *Computational Complexity*

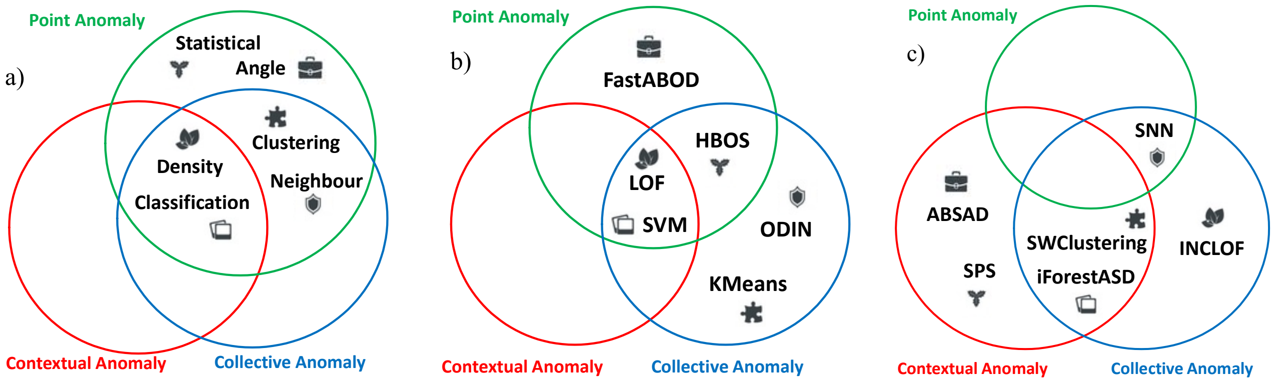
Despite not being the focus of our analysis, for completeness we report here on the computational complexity of the algorithms we select for our study. The complexity of *density-based* and *angle-based* algorithms is at least O(N^2), or rather the complexity of an NN query. *Clustering* is generally more efficient, with complexity of O(k N), where *k* represents the number of clusters. S*tatistical* and *classification* families may have very different complexities; therefore it is not easy to present a bound. However, statistical algorithms are usually sub-linear e.g., O(N log N) for *HBOS*, while classifiers usually build complex structures e.g., tree ensemble [25], or conduct complex calculations e.g., *OneClassSVM*, O(N^2).

C. *Final Discussion*

Overall, our results show which unsupervised algorithms – and their families – are more suitable than others to detect anomalies belonging to specific classes. The final results of our qualitative and quantitative analyses are depicted in Figure 4. More in detail, Figure 4-A reports on the qualitative analyses discussed in Section IV, while Figure 4-B and 4-C reports on the capabilities of non-sliding and sliding window algorithms with respect to the three known classes of anomalies. Figure 4-B and 4-C are built considering non-sliding algorithms as "capable" of identifying a class of anomalies if in our experiments a given algorithm reached or exceeded 0.8 of MCC. We lower this threshold to 0.65 for sliding window algorithms. We choose these two thresholds trying to balance the "good" and the "bad" i.e., a MCC of 0.65 splits the scores of Figure 2 in two groups with the same cardinality.

By looking at Figure 4 as a whole, we can notice several mismatches e.g., Figure 4-A shows *clustering* algorithms as capable of identifying both point and collective anomalies, while Figure 4-B puts *KMeans*, the non-sliding clustering algorithm, in the "collective anomaly" set. Discrepancies are mainly related to the specific implementations of algorithms: despite they clearly belong to a given family, they may use some specific techniques that slightly change their behavior. Instead, LOF and SVM scores confirm the

Fig. 4. Linking Classes of Anomalies to a) Families of Algorithms, b) Non-Sliding Algorithms, and c) Sliding Window Algorithms. Symbols match an algorithm with its family in the pictures.



results of qualitative analyses on density-based and classification families i.e., they fall in the intersection of the three sets in Figure 4-A and Figure 4-B. Unfortunately, as reported in Section VI.B, they both have quadratic computational complexity, meaning that the time they need for training and for providing an anomaly score for a novel data point is not optimal. However, faster algorithms such as *clustering* and *statistical* (sub-linear complexity), have worse detection capabilities, as showed by our metric scores. A possible solution could be the parallelization of different fast algorithms, with a final voting of their results.

Figure 4-C deserves a specific mention. The usage of sliding windows makes difficult to define a global notion of expected behavior, and therefore the ability of detecting point anomalies is affected. In fact, only the SNN algorithm has a MCC score higher than 0.65 when aiming at point anomalies. Instead, algorithms belonging to families supposed to be specifically suitable for point anomalies i.e., *angle-based* and *statistical*, show good performances in detecting contextual anomalies. These algorithms identify point anomalies in the restricted context described by the sliding window, which can be paired with contextual anomalies. Other sliding window algorithms follow the same pattern being effective in identifying contextual anomalies. As a final remark, we do not identify a generalist sliding window algorithm e.g., effective in identifying all classes of anomalies with good scores.

D. Our Main Findings

As a final recap, we summarize our findings as follows.

- *Contextual anomalies are not easily detectable by non-sliding algorithms*, since they process the training data all-at-once, making the identification of local contexts more difficult.
- *Sliding window algorithms generally struggle in identifying point anomalies*, since the restricted amount of data points in the window does not allow building a global knowledge.
- We observe how *an accurate management of data points in the sliding window* i.e., avoid putting anomalous data points in the window, *improves the overall scores of sliding window algorithms.*
- *SVMs turned out to be the best overall algorithm.* However, its training phase is heavy i.e., O(N^2), and strongly dependent on several assumptions which may cause the definition of the boundary to fail.
- *Running different algorithms in parallel and combining their scores may provide good detection capabilities*, also allowing to manage the overall complexity by selecting the faster ones.
- *There is no anomaly threshold* that turned out to be overall more useful to improve detection capabilities of algorithms with our datasets.

VII. Conclusions and Future Works

We investigated the suitability of unsupervised algorithms and their families in detecting either *point*, *contextual* or *collective* anomalies. We first reviewed the literature of unsupervised anomaly detection algorithms, devising initial conjectures on their suitability to classes of anomalies through qualitative analyses. Then, the initial analyses were substantiated by quantitative analyses conducted by executing the selected algorithms on three datasets, reporting results throughout common scoring metrics. Our study involved 12 unsupervised algorithms, a non-sliding and a sliding window algorithm for each of the 6 (*clustering*, *statistical*, *classification*, *neighbour-based*, *density-based* and *angle-based*) families of algorithms. Results showed that non-sliding algorithms effectively identify point and collective anomalies, while sliding window algorithms are particularly suitable for contextual anomalies. Additional side discussions are expanded and then summarized at the end of the paper.

As future work, we will investigate and evaluate if and how combining different algorithms helps improving detection capabilities, along with a sensitivity analysis directed to discover optimal anomaly thresholds. Moreover, mixtures of both sliding and non-sliding algorithms may reveal to be particularly suitable for specific needs or systems. In particular, if a system partially changes its behaviour and a novel training phase for non-sliding algorithms is triggered, sliding window algorithms may be employed as a temporary solution until the training phase involving the main detection algorithm completes.

Acknowledgment

This work has been partially supported by the REGIONE TOSCANA POR FESR 2014-2020 SISTER and FAR-FAS 2014 TOSCA-FI projects.

REFERENCES

[1] J. T. Force and T. Initiative, “Security and privacy controls for federal information systems and organizations”, NIST Special Publication, vol.800, no. 53, pp. 8–13, 2013.

[2] Chandola, V., Banerjee, A., & Kumar, V. (2009). “Anomaly detection: A survey”. ACM computing surveys (CSUR), 41(3), 15.

[3] Bontemps, L., McDermott, J., & Le-Khac, N. A. (2016, November). Collective anomaly detection based on long short-term memory recurrent neural networks. In International Conference on Future Data and Security Engineering (pp. 141-152). Springer, Cham.

[4] Zheng, Y., Zhang, H., & Yu, Y. (2015, November). “Detecting collective anomalies from multiple spatio-temporal datasets across different domains”. In Proceedings of 23rd ACM SIGSPATIAL Int. Conference on Advances in Geographic Information Systems (p. 2).

[5] Modi, Chirag, et al. "A survey of intrusion detection techniques in cloud." Journal of Network and Computer Appl. 36.1 (2013): 42-57.

[6] Gu, Zhongshu, Kexin Pei, Qifan Wang, Luo Si, Xiangyu Zhang, and Dongyan Xu. "LEAPS: Detecting camouflaged attacks with statistical learning guided by program analysis." In Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on, pp. 57-68. IEEE, 2015.

[7] Goldstein, Markus, and Andreas Dengel. "Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm." KI-2012: Poster and Demo Track (2012): 59-63.

[8] Salfner, Felix, Maren Lenk, and Miroslaw Malek. "A survey of online failure prediction methods." ACM Computing Surveys (CSUR) 42.3 (2010): 10.

[9] Kriegel H-P, Zimek A. “Angle-based outlier detection in high-dimensional data”. In: Proceedings of the 14th ACM SIGKDD Int. Conference on Knowledge discovery and data mining; ‘08. p. 444–52.

[10] M.Goldstein and S.Uchida, “A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data,” PloS one, vol. 11,no. 4, p.e 152 - 173, 2016.

[11] Hautamaki, V., Karkkainen, I., & Franti, P. (2004, August). Outlier detection using k-nearest neighbour graph. In Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on (Vol. 3, pp. 430-433). IEEE.

[12] Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. computers & security, 28(1-2), 18-28.

[13] Liao, Y., & Vemuri, V. R. (2002). Use of k-nearest neighbor classifier for intrusion detection1. Computers & security, 21(5), 439-448.

[14] B. Scholkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C.Williamson, “Estimating the support of a high-dimensional distribution”, Neural computation, vol.13, no.7, pp. 1443–1471, 2001.

[15] M. Amer, M. Goldstein, and S. Abdennadher, “Enhancing one-class support vector machines for unsupervised anomaly detection,” in Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description. ACM, 2013, pp. 8–15.

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

[17] Schubert, E., Koos, A., Emrich, T., Züfle, A., Schmid, K. A., & Zimek, A. (2015). A framework for clustering uncertain data. Proceedings of the VLDB Endowment, 8(12), 1976-1979.

[18] Zhou, A., Cao, F., Qian, W., & Jin, C. (2008). Tracking clusters in evolving data streams over sliding windows. Knowledge and Information Systems, 15(2), 181-214.

[19] Karimian, S. H., Kelarestaghi, M., & Hashemi, S. (2012, May). I-inclof: improved incremental local outlier detection for data streams. In Artificial Intelligence and Signal Processing (AISP), 2012 16th CSI International Symposium on (pp. 023-028). IEEE.

[20] Bondavalli, A., Brancati, F., & Ceccarelli, A. (2009, October). “Safe estimation of time uncertainty of local clocks”. In Proc. of Int. IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication, ISPCS (pp. 47-52).

[21] Zoppi, T., Ceccarelli, A., & Bondavalli, A. (2016, September). Context-Awareness to Improve Anomaly Detection in Dynamic Service Oriented Architectures. In Int. Conference on Computer Safety, Reliability, and Security (pp. 145-158). Springer, Cham.

[22] Zhang, Liangwei, Jing Lin, and Ramin Karim. "Sliding window-based fault detection from high-dimensional data streams." IEEE Transactions on Systems, Man, and Cybernetics: Systems 47.2 (2017): 289-303.

[23] Mouratidis, K., & Papadias, D. (2007). Continuous nearest neighbor queries over sliding windows. IEEE transactions on knowledge and data engineering, 19(6), 789-803.

[24] Ding, Z., & Fei, M. (2013). An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. IFAC Proceedings Volumes, 46(20), 12-17.

[25] Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008, December). Isolation forest. In Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on (pp. 413-422). IEEE.

[26] N. Moustafa and J. Slay, “UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set),” in Military Communications and Information Systems Conference (Mil-CIS), 2015. IEEE, 2015, pp. 1–6.

[27] Zoppi, T., Ceccarelli, A., & Bondavalli, A. (2017, April). Exploring anomaly detection in systems of systems. In Proceedings of the Symposium on Applied Computing (pp. 1139-1146). ACM.

[28] Tang, J., Chen, Z., Fu, A. W. C., & Cheung, D. W. (2002, May). Enhancing effectiveness of outlier detections for low density patterns. In Pacific-Asia Conference on Knowledge Discovery and Data Mining (pp. 535-548). Springer, Berlin, Heidelberg.

[29] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the kdd cup 99 data set,” in Computational Intelligence for Securit and Defense Applications, 2009. CISDA 2009. IEEESymposium on. IEEE, 2009, pp. 1–6

[30] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, “Toward developing a systematic approach to generate benchmark datasets for intrusion detection,”computers & security, vol. 31, no. 3, pp. 357–374, 2012.

[31] G. O. Campos, A. Zimek, J. Sander, R. J. Campello, B. Micenkova, E. Schubert, I. Assent, and M. E. Houle, “On the evaluation of outlier detection: Measures, datasets, and an empirical study”, in Lernen, Wissen, Daten, Analysen 2016.ceur workshop proceedings, 2016.

[32] Boughorbel, Sabri, Fethi Jarray, and Mohammed El-Anbari. "Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric." PloS one 12.6 (2017): e0177678.

[33] J. F ̈urnkranz and P. A. Flach, “Roc nrule learning towards a better understanding of covering algorithms”, Machine Learning, vol. 58,no. 1, pp. 39–77, 2005.

[34] D. M. Powers, “Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation,” 2011

[35] Source Files (online), https://github.com/tommyippoz/Miscellaneous-Files/blob/master/PRDC_Submission_Datasets_Scores.rar, accessed: 2018-05-30

[36] “Elki data mining,” elki-project.github.io, accessed: 2018-05-30

[37] “Weka 3: Data Mining Software in Java”, https://www.cs.waikato.ac.nz/~ml/weka/, accessed: 2018-05-30

[38] Benesty, J., Chen, J., Huang, Y., & Cohen, I. (2009). Pearson correlation coefficient. In Noise reduction in speech processing (pp. 1-4). Springer Berlin Heidelberg.

[39] Chicco, Davide. "Ten quick tips for machine learning in computational biology." BioData mining 10.1 (2017): 35.

[40] Leung, K., & Leckie, C. (2005, January). Unsupervised anomaly detection in network intrusion detection using clusters. In Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38 (pp. 333-342). Australian Computer Society, Inc..

[41] E. Curry, S. Hasan, N. Pavlopoulou, T. Zaarour et al., “Grand challenge: Automatic anomaly detection over sliding windows,” in Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems. ACM, 2017.

[42] L. Zhang, J. Lin, and R. Karim, “Sliding window-based fault detection from high-dimensional data streams”, IEEE Transactions on Systems, Man, and Cybernetics, vol. 47, no. 2, pp. 289–303, 2017.

[43] He, S., Zhu, J., He, P., & Lyu, M. R. (2016, October). Experience report: system log analysis for anomaly detection. In Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on (pp. 207-218). IEEE.

[44] Lin, Q., Zhang, H., Lou, J. G., Zhang, Y., & Chen, X. (2016, May). Log clustering based problem identification for online service systems. In Proceedings of the 38th International Conference on Software Engineering Companion (pp. 102-111). ACM.