Computer Communications 202 (2023) 191–203

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Anomaly detection for fault detection in wireless community networks using machine learning

Llorenç Cerdà-Alabern [a,*], Gabriel Iuhasz [b], Gabriele Gemmi [a,c]

[a] Universitat Politècnica de Catalunya, Barcelona, Spain
[b] West University, Timisoara, Romania
[c] University of Venice Ca' Foscari, Italy

ARTICLE INFO

Keywords:
Fault detection
Anomaly detection
Machine learning
Wireless network dataset
Wireless community networks

ABSTRACT

Machine learning has received increasing attention in computer science in recent years and many types of methods have been proposed. In computer networks, little attention has been paid to the use of ML for fault detection, the main reason being the lack of datasets. This is motivated by the reluctance of network operators to share data about their infrastructure and network failures. In this paper, we attempt to fill this gap using anomaly detection techniques to discern hardware failure events in wireless community networks. For this purpose we use 4 unsupervised machine learning, ML, approaches based on different principles. We have built a dataset from a production wireless community network, gathering traffic and non-traffic features, e.g. CPU and memory. For the numerical analysis we investigated the ability of the different ML approaches to detect an unprovoked gateway failure that occurred during data collection. Our numerical results show that all the tested approaches improve to detect the gateway failure when non-traffic features are also considered. We see that, when properly tuned, all ML methods are effective to detect the failure. Nonetheless, using decision boundaries and other analysis techniques we observe significant different behavior among the ML methods.

1. Introduction

Anomaly detection (AD) is aimed at identifying deviations from the expected behavior [1]. The identification of these unusual behaviors has proven to be a powerful tool in a wide range of applied sciences. Some examples are credit-card fraud detection, medical diagnosis, industrial processes, and computer networks [2]. Although different methods can be applied to AD [3], recent advances in ML methods and their ability to learn from data have boosted the number of AD proposals using ML techniques [4].

Machine learning uses a dataset to make probabilistic predictions [5,6]. A group of dataset samples, referred to as training set, is used to train the algorithm. Then, predictions are made over a different group of samples referred to as the testing set. In some applications the training set is labeled. For instance, in optical character recognition the training set consists of images and the labels are their corresponding characters. These problems are called supervised learning. In contrast, in AD an unlabeled dataset is usually used, and ML algorithms are called unsupervised. In AD the training set is simply used as a representation of the expected normal operation, and anomalies are identified by deviations from the expected behavior.

In the field of computer networks, AD has been primarily concerned with security, especially network intrusion detection [3,7,8]. In contrast, in this paper we focus on fault detection, another essential need in computer networks where AD can also be of great interest, but which has received little attention. Some difficulties explain the few works that can be found in the literature on AD for fault detection. The main one is the lack of datasets. The reason is that a realistic dataset for fault detection would consist of features related to network traffic and hardware metrics, as CPU load and memory usage. This kind of dataset would be difficult to generate by simulation, but should be obtained from a real testbed or, ideally, from a production network. However, for confidentiality reasons commercial network operators do not make these types of datasets of their networks public.

Another goal of our work is therefore to create a dataset with real data. To do so we will focus on AD in Wireless Community Network (WCN). Community networking, also known as bottom-up networking, is a model that emerged by the need for broad Internet access in under-provisioned zones, typically rural areas and developing countries. Nowadays there are hundreds of community networks operating in very diverse ways [9]. WCNs are non-profit networks built by their own users, normally installing wireless antennas on top of their houses. Sometimes WCNs users do not only build the network infrastructure, but associate to constitute micro ISPs, providing access to the Internet and internal services managed by the community. WCNs are similar to Wireless Internet Service Providers (WISPs) [10] — typically small

* Corresponding author.
E-mail address: llorenc.cerda@upc.edu (L. Cerdà-Alabern).

https://doi.org/10.1016/j.comcom.2023.02.019
Received 20 September 2022; Received in revised form 18 January 2023; Accepted 20 February 2023
Available online 23 February 2023

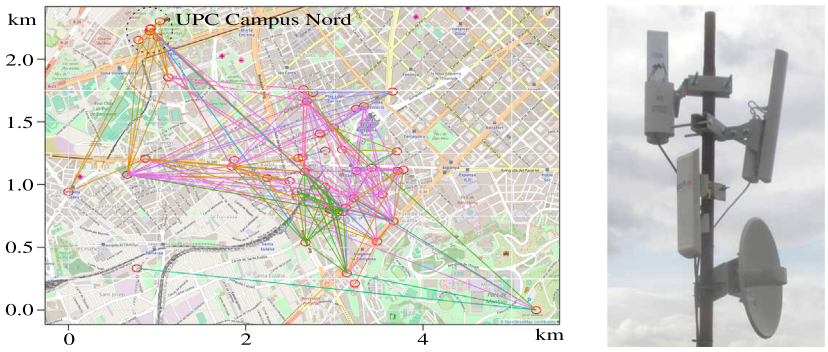


Fig. 1. GuifiSants topology. Colors indicate links configured in the same WiFi channel, and outdoor routers.

businesses with reduced margin profits and the aim to provide broad Internet to their community.

The non-profit and shared resources of community networks typically make their users open to participate in research projects, and provide access to data from their network infrastructure. We have thus focused on WCN with the objective of building a dataset of a production network. In particular we have considered Guifi.net. Guifi.net is one outstanding WCN example. It started in 2004 and, at the time of writing, reports 36.886 working nodes [11]. Guifi.net has become a complex network where associations of self-providing users coexist with commercial network operators [12].

In this paper, we analyze a dataset we have gathered from a production WCN which is part of Guifi.net. This WCN has been deployed in a quarter of Barcelona, Spain, and it is called GuifiSants [13]. When the dataset was gathered there were around 60 nodes. GuifiSants is highly heterogeneous in its topology, quality of the links, and node behavior. There are nodes located at higher buildings having several antennas, some sector and other parabolic creating high capacity point-to-point links. These nodes form a kind of unplanned backbone. Other non-technical users have end nodes made of a single antenna. Fig. 1 shows the geographic locations of the GuifiSants nodes, and a picture of the outdoor routers deployed in one node. Depending on the antenna types, obstruction of the links, and 802.11n/ac WiFi technologies used, the link capacity varies from a few Mbps to several hundreds. The stability of the nodes is also rather variable. For instance, there are users that frequently update the software, or reboot their node if the connection is not working satisfactorily. Backbone nodes are more stable and have few failures or reboots. Being a mesh network, the connectivity is rather resilient: even if a backbone node fails, the routing protocol re-configures alternative routes and most nodes can still reach the Internet, which is the main service consumed by the users. AD in such a WCN is challenging due to its heterogeneous and diverse nature. Nevertheless, we believe it is not only an interesting study in its own right, but other scenarios as large IoT networks may share similar diversity features, allowing us to extrapolate some of the lessons learned from WCN analysis.

Most AD studies related to computer networks found in the literature use traffic-related features [14–16]. In contrast, the dataset we have produced contains not only traffic, but features from the Linux kernel that reflect e.g. the state of the CPU and memory. Such features are typically gathered by network monitoring tools such as Munin [17] and Nagios [18]. These tools, however, simply produce time-series graphics of the gathered features, and their analysis and interpretation are left to the network administrator. We are thus interested in whether including these additional features to the basic ones based on traffic can increase the AD capability.

Summing up, our main contributions in this paper are the following:

- We have produced a dataset gathering a large set of features related not only to traffic, but other parameters such as CPU and memory. The dataset has been gathered from a production WCN and we have made it available in the public Zenodo repository [19]. To our best knowledge, this is the first dataset containing such a rich diversity of features obtained from a production wireless network available in a public repository.
- We have used AD methods using the dataset to perform a fault detection analysis using 4 ML unsupervised learning approaches based on different principles. We investigate the performance of these well-known ML methods applied to AD in wireless networks and their behavior by varying some system parameters, such as the size of the dataset.

The rest of the paper is organized as follows: Section 2 provides some related works. Section 3 summarizes the 4 ML approaches used in this paper. Section 3 describes the methodology and features gathered in the dataset under study. Section 5 provides the numerical results and Section 6 concludes the paper.

2. Related work

Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior [1]. In statistics, the term outlier is often used for this concept. The importance of AD is due to the fact that anomalies in data might reveal critical misbehavior in a wide number of scenarios. For instance, fraud detection for credit cards, insurance or health care, damage detection, etc. The significance of this topic has motivated a large number of articles, surveys, and even books. For instance, the surveys [1,20] provide an extensive review of detection techniques developed in a wide number of domains.

In computer networks, most work on AD has traditionally focused on the problem of security. In this context the term *Intrusion Detection System* (IDS) is often used. IDSs typically exploit anomalies in network traffic to detect a wide number of security attacks, such as denial of service (DoS), reconnaissance, etc. See e.g. [21] for a description of different types of security attacks. An overview of IDSs proposed in the literature can be found in [7]. In [22], PCA (Principal Component Analysis) was introduced to perform a structural analysis of network traffic. Their proposal received significant attention for IDS. For instance, in [23,24] refinements to PCA analysis are done to detect spikes in traffic flows, which potentially identify network attacks. In [25] generate port-scans and snapshots attacks in a lab, merged with residential traffic. Attacks detection is done using a robust PCA approach. PCA to detect network traffic anomalies has also been criticized by a number of papers [26]. In [27] it is analyzed the weaknesses of PCA, and it is used *Commute Distance* to detect DoS attacks using traffic measures. In [14] it is argued that problems with PCA are due to flaws in its adoption, and appropriate ways to apply PCA are proposed.

ML methods have a long tradition of being used for AD. Until recently most research has been done on using supervised ML methods coupled with synthetic or heavily annotated datasets [28]. Recently this trend has shifted towards utilizing real-life datasets coupled with unsupervised methods. Even so, there are not many research articles

that have both a comparison of unsupervised AD methods coupled with a freely available real-life dataset.

A very good overview of unsupervised methods on several datasets is given in [29]. Here we see the performance of several well-known AD methods such as k-NN, CBLOF, etc. More relatable research including some of the same methods has been done in [30] where authors compare several clustering-based AD techniques on network traffic data. Other types of AD methods have also been used in network management systems with high dimensional network data [31].

Deep learning based methods are also utilized for AD. In [32] the authors provide a comprehensive overview of different deep learning methods for AD including unsupervised methods. They also correctly point out that explainability is a key issue, especially when using AD methods which are traditionally considered as black-box models. The problem of explainability is also discussed in [33,34] in the case of different deep learning models, which include Autoencoders [35] and Variational-Autoencoders [36] respectively.

Most of the research done regarding unsupervised methods for AD is largely focused on network intrusion detection. In [37] there is one of the few works found in the literature where network AD is investigated using ML with a real dataset. However, in [37] the authors use AD to study the behavior of TCP traces to investigate the performance of a 4G cellular network, which is quite a different scenario than ours. The methods used in the literature can be divided into several types. These include clustering based methods [38–41], outlier based techniques [37,42–45] and soft computing based techniques [46–48]. All of the methods described here have both pros and cons in addition to the problem of explainability already mentioned before. Clustering based methods usually can handle only continuous features, inappropriate proximity measures adversely affect the detection rate. Most outlier based methods are complex (typically used in conjunction with clustering) and are highly parameter dependent, while soft computing based methods usually require large quantities of historical data with overfitting of models being arguably a greater issue than for supervised methods. In essence the selection of which type of method and what training parameters to use is very much dependent on the problem domain and the available data.

3. Machine learning based approaches

In the following section, we will focus on testing several unsupervised ML techniques for AD. For ML techniques AD is split up into several categories based on the type of methods and the characteristics of the available data. The simplest form of anomalies are point anomalies which can be characterized by only one feature making them easier to detect. Other types of anomalies are more complex but ultimately yield a much deeper understanding of the inner workings of a monitored system and/or application [1]. These types of anomalies are fairly common in complex geographically distributed systems.

Contextual anomalies are extremely interesting in the case of complex systems. These types of anomalies happen when a certain pattern of feature values is encountered. In isolation, these values are not anomalous but when viewed in context they represent an anomalous event. These types of anomalies can represent application bottlenecks, imminent hardware failure, software misconfiguration, or even malicious activity. The last major types of anomalies, which are relevant, are temporal and sequential anomalies where a certain event takes place out of order or at the incorrect times. These types of anomalies are very important in systems that have a strong spatial–temporal relationship between features, which is very much the case in dynamic distributed systems such as mesh networks.

One important consideration for most AD tasks is the fact that labeled high-quality data is rarely available. Moreover, supervised methods trained on labeled data will not be able to detect new or unforeseen anomalous events. Unsupervised methods do not have this limitation, however, these methods have several downsides. First, they are in many cases prone to a high false-positive rate. This can be due to many factors such as overfitting, large feature spaces, etc. Second, even if anomalies are detected the algorithms themselves usually are unable to give any meaningful insight into what type of anomaly has been detected or even what caused the anomalous event. In this section of our paper, we will focus on alleviating some of the issues presented here in the case of unsupervised detection methods. Note that we shall refer as *sample* each measurement of the dataset. Thus, we shall use the terms *anomaly event*, *anomaly sample* or simply *anomaly*, interchangeably.

3.1. Principal component analysis

Principal Component Analysis, PCA, is a standard methodology applied to Statistical Process Control [14]. The idea of PCA is approximating the samples of n features by a projection on a lower dimensional space of dimension $l < n$. PCA is well studied in the literature, so we omit the description of this method and only discuss the statistics we used for our analysis. The interested reader can refer to [14] for further details.

In order to detect anomalies of a new score there are several indices [14,49]. We have found that in our study the best performing one is the squared prediction error (SPE), also known as the Q statistic:

$$Q_j = \sum_{i=1}^{n} e_{ji}^2 \tag{1}$$

where e_{ji} the features' ith residual of sample j.

Once an anomaly is detected, a diagnosis system is needed to determine its root causes. The general approach is using contribution plots [50]. The idea of such plots is estimating the contribution of each observed feature of a sample to a particular statistic value considered anomalous. There have been proposed several methods to build contribution plots [49]. In this paper, we shall use complete decomposition contributions (CDC). CDC are easily computed and show a good diagnosis performance in a networking monitoring system [14]. The CDC for the Q statistics defined above is given by [49]:

$$CDC_i^{Q_j} = e_{ji}^2. \tag{2}$$

3.2. Isolation Forest

In recent years, ensemble based detectors have garnered popularity in the case of AD. These methods combine the outputs of multiple algorithms called base detectors which are then used to create a unified output. These algorithms operate on the assumption that some algorithms perform well on a subset of the available data while others can perform better on a different subset. Ensemble combination on the other hand often outperforms individual estimators because of their ability to combine the outputs of multiple algorithms. Most ensemble methods require labeled data however, it has been shown both from a theoretical [51] and practical [52] point of view that unsupervised ensemble methods have characteristics with their supervised counterparts. We can formulate a modified bias–variance trade-off for the anomaly analysis setting [51], this will enable many of the supervised algorithms to be "generalized" into unsupervised tasks. One of the key issues which is caused as a direct result of this generalization is the increased difficulty in selecting appropriate hyper-parameters. We should also note that ensemble models are useful when dealing with difficult algorithmic choices by reducing the uncertainty inherent in these.

Isolation Forest is an outlier ensemble based algorithm which is constructed from multiple isolation trees [53]. It explores random subspaces from the data. In essence, it explores random local subspaces as each tree uses different splits. Scoring is done by qualifying how easy it is to find a local subspace of low dimensionality in which a particular event is isolated [54]. In other words distance from the leaf to the root is used as the outlier score. Similar to Random Forest a supervised

method, the final score is obtained by averaging the path length of any particular data point in different isolation trees. In most scenarios, Isolation Forest works under the assumption that it is more likely to detect or isolate an outlier in a subspace of lower dimensionality created by the random splits.

During the training phase, Isolation Forest constructs the unsupervised equivalent of decision trees. These trees are binary and have at most N leaf nodes which is equal to the number of data points in the training set. Thus the root node contains all of the data points to be processed and serves as the initial state of the isolation tree T. Next, a candidate list C is initialized containing the root node. From this list, a node denoted as R is randomly chosen. A randomly chosen attribute i is used to split R into two sets R_1 and R_2. This is done by selecting a random value a from attribute i such that all data points from R_1 satisfy $x_i \leq a$ and all data points in R_2 satisfy $x_i > a$. This value a is chosen uniformly at random between the minimum and maximum values of the ith attribute among the data points in node R. Both R_1 and R_2 are children of R in T. In case R_i contains more than one point then we add it in C otherwise we designate the node as a leaf. We repeat this process until C is empty.

We can see that the result of training will yield a non-balanced binary tree and that outlier nodes will typically be isolated at a lower dimensionality than normal points. The inherently randomized approach is repeated multiple times and the results are averaged (ensembled) with a computational complexity of $\Theta(N \log(N))$ and space complexity of $O(N)$. Isolation trees create hierarchical clusters from the data, however, they are extremely randomized as can be seen from the previous paragraphs. This leads to some issues when using this method on real-world datasets. By using subsampling it is possible to improve computational performance and aid in creating diversity which is crucial for ensemble methods [52]. However, diversity is also gained through the randomized process of isolation tree construction, arguably even more so than subsampling. Thus, the correct selection of features or attribute i is very heavily influenced by randomness. It is entirely possible to have poor-performing models because of poor feature selection during training.

3.3. Clustering-based local outlier factor

There are several underlying assumptions that stand at the basis of unsupervised methods. Proximity-based techniques define an anomalous event (or data point) when its locality is sparsely populated. This proximity can be defined in several ways. In cluster-based methods non-membership of an event relative to a cluster is its distance from other clusters, the size of the closest cluster, or a combination of these factors, are used to calculate an anomaly score. Some methods are heavily cluster based, basically anything which cannot be assigned to a cluster is considered an anomaly.

Distance-based methods base their anomaly score around the distance of an event to its k-nearest neighbors. In this case events with high k-nearest neighbors distances as anomalous. Distance-based methods usually have a greater granularity which comes with an increased computational cost. Density-based methods work based on the number of events that are in a local region, this is then used to define local density and compute the anomaly score. Basically, density-based methods partition data space while clustering-based methods partition data points.

Clustering-Based Local Outlier Factor (CBLOF) [55] is a proximity based algorithm, which is a combination of local outlier factor (LOF) and a clustering technique [56]. LOF adjusts the anomaly (or outlier) score based on local density. The density is defined as an inverse of average distances. This approach results in events in local regions of high density being given higher anomaly scores even if they are isolated from the other events in their locality. This is mainly due to the definition of density which does not contain the number of anomalies in a cluster. In fact, CBLOF is a score in which anomalies are defined as a combination of local distance to nearby clusters and the size of the clusters to which each event belongs. Thus events in small clusters that are at large distances to nearby clusters are flagged as anomalies.

3.4. Variational AutoEncoders

Variational AutoEncoders (VAE) are deep neural network models designed for unsupervised training, which can be used for AD tasks [57,58]. They are often mentioned together with Autoencoders (AE) which are also deep learning models with seemingly similar topological components: encoder and decoder. The encoder tries to learn a lower-dimensional representation of the input data (similar to PCA) and a decoder that attempts to reproduce the input data in the original dimension (AE are usually symmetrical). AEs try to encode the data in such a way that they reduce the reconstruction error. When used for AD AEs reconstruction error can be used as a form of anomaly score [59]. Provided the AE has sufficient training data to provide a minimal reconstruction error for normal data.

VAEs on the other hand try to encode the data into a multivariate latent distribution which is then used by the decoder. The difference between AE and VAE is that instead of generating a latent vector, which the decoder can reproduce, VAEs learn two vectors that represent the mean and variance parameters of a distribution from which the latent vector is sampled and used by the decoder function for reconstructing the original input. In general latent space generated by VAEs tend towards normality due, in no small part, to the fact that encoders are heavily regularized (via Kullback–Leibler divergence term).

VAEs have some issues in common with other deep learning models. First, it is relatively slow to train, often requiring specialized hardware such as GPGPU. Large datasets also require complex network topologies. However, if complex topologies are applied to small datasets this can lead to overfitting. One regularization method which has been shown to work in reducing overfitting is the addition of dropout layer [60].

A variant of VAE called $\beta - VAE$ has been proposed for anomaly detection [61,62]. This variant ads a new hyper-parameter called β which constricts the encoding capacity of the latent bottleneck and encourages latent representations to be factorized. It should be noted that the standard VAE can be considered a special case of this new variant, the two being equivalent when $\beta = 1$.

3.5. Feature selection

High dimensional unbalanced datasets are one of the main challenges of ML research field [63]. By using methods that enable the manual or automatic choosing of a relevant subset from an initially large and potentially redundant set of initial features we can potentially improve both the accuracy as well as the computational overhead of ML methods. A dataset that has a lower dimensionality and is well balanced can also substantially limit overfitting and provide an easier interpretation of prediction results [64]. In the case of unsupervised feature selection, one cannot rely on target variables but instead, utilize some other target criteria and clustering to improve detection accuracy.

The Kurtosis measure can be calculated on each feature as a form of unsupervised feature selection method. The Kurtosis of a feature f_i, $i = 1, \ldots, n$ of m samples is computed by first standardizing its samples with zero mean and unit standard deviation:

$$k_j = \frac{f_{ij} - \mu_i}{\sigma_i}, \; j = 1, \ldots, m \tag{3}$$

Where μ_i represents the mean and σ_i the standard deviation of f_i, and f_{ij} the jth sample of ith feature. Next, the Kurtosis of f_i is computed as:

$$K_i = \frac{\sum_{j=1}^{m} k_j^4}{m} \tag{4}$$

Kurtosis is the measure of "non-uniformity" of the data. It describes the shape of the probability distribution. A high Kurtosis value usually corresponds to a high degree of deviation/outliers although the interpretation is variable based on the context it is applied on. In our case, the Kurtosis level calculated for each feature allows us to select only

those features which have a relatively high non-uniform distribution. A known downside of the Kurtosis measure is that it does not consider interactions between various features. Multidimensional Kurtosis can be computed by computing the Kurtosis on the Mahalanobis distance of all data points to the centroid of the reprojection of the original data into a lower-dimensional subspace.

Mean Absolute difference computes the absolute difference from the mean value. A higher score denotes a greater discriminatory potential [65]. It is defined as:

$$MAD_i = \frac{1}{m}\sum_{j=1}^{m}|f_{ij} - \mu_i| \tag{5}$$

3.5.1. Shapely values

The above-mentioned methods are a useful way of identifying features that might yield a better-performing predictive model. One interesting new approach, not only to select features with a high degree of impact on a prediction but also to explain, in the case of unsupervised methods, why a particular event has been deemed as anomalous is Shapely values [66]. These have been first introduced in the study of coalition games. It is defined on a value function denoted as v of players in S. The Shapely value denotes the contribution to the payout of a particular feature value, weighted and summed over all possible combinations:

$$\phi_i(v) = \sum_{S\subseteq\{1,\dots,n\}\setminus\{i\}} \frac{|S|!\,(n-|S|-1)!}{n!}(v(S\cup\{i\}) - v(S)) \tag{6}$$

where n denotes the set of all players thus the Shapely value of the game (v, n) is used to distribute the total gain $v(n)$ to each player in accordance with each contribution [33]. In the context of machine learning in general and AD in particular a player i corresponds to a feature from the dataset, this way n denotes the total number of input features. Conversely, the Shapely value of a feature $i \in n$, $\phi_i(v)$ is the weighted average of the marginal contribution. Resulting from Eq. (6) we can compute the prediction for feature values in set S, a subset of the features used to train the model, which are marginalized over features that are not in set S as follows [67]:

$$v_x(S) = \int f(x_1, \dots, x_n)\, dP(x \notin S) - \mathrm{E}_X[f(X)] \tag{7}$$

In the context of ML models X is the vector of values from the features of the instance to be explained and n the number of features. Shapely values are symmetrical in the sense that equal contribution results in equal Shapely values and non-contributing features have a shapely value of 0. It also has the efficiency property, meaning that feature contributions sum up to the difference of prediction for x and the average prediction.

The main issue for computing Shapely values for feature selection is that they require a labeled dataset or prediction to explain. In our case, we have unsupervised methods we wish first to have an explanation of why a particular event has been identified as anomalous. This use-case fits very well with Shapely Values. However, we can also compute the feature importance based on the anomalous events detected and the Shapely values calculated. This step can be used to reduce the dataset feature space considerably while at the same time it has the potential to increase the anomalous event detection rate.

4. Dataset

In this paper, we will focus on a WCN called GuifiSants [13]. GuifiSants started in 2009 and is part of Guifi.net. The nodes of GuifiSants consist of antennas flashed with a Linux Openwrt distribution [68] running the BMX6 mesh routing protocol [69,70]. GuifiSants is a WCN deployed in a neighborhood of the city of Barcelona (Spain) called Sants. In 2012 *Universitat Politècnica de Catalunya* (UPC) joined GuifiSants for research purposes. At the time of writing, there are around 60 nodes in GuifiSants. GuifiSants is a production network with some tens of users having this network as their only access to the Internet. In [71] a live monitoring web page of GuifiSants updated hourly is available. Technical analysis of GuifiSants can be found in [72].

The dataset has been built from data samples gathered from each node every 5-minutes. The dataset is publicly available in Zenodo [19]. This is done using a permanent ssh connection from one central monitoring server to each node in the mesh, which is used to run standard system commands. The dump of these commands is then parsed to obtain the data. This method has the advantage that no changes or additional software need to be installed in the nodes. This is an important condition since the users are the owner of their nodes. Therefore, only the users' permission is needed to install a public key to access the node with ssh for monitoring purposes.

The data is obtained by reading the Linux kernel variables available through the `/proc` file-system. For instance, `/proc/net/dev` to read counters with the number of bytes and packets transmitted and received over each interface; `/proc/stat` where there is information about kernel activity; `/proc/meminfo` for memory usage, etc. Kernel variables are of two types: (i) absolute values, for instance, the CPU 1-minute load average, and (ii) counters that are monotonically increased, for instance, the number of transmitted packets. We have converted the counter-type kernel variables into rates by dividing the difference between two consecutive samples over the difference of the corresponding timestamps in seconds. We have eliminated the negative rate samples that occur when a node reboots, or when a counter reaches its maximum value and restarts.

The dataset consists of traffic and non-traffic features. Traffic features are obtained from the counters available at `/proc/net/dev` Linux /proc file-system. For each node, we have considered the counters of received and transmitted bytes and packets over Ethernet and WiFi interfaces.

For each node, we have also considered the sum of counter values for both types of interfaces (Ethernet and WiFi), and the sum and difference of received and transmitted bytes and packets over all interfaces. Recall that all counter features are then converted into rates, as explained above. For instance, features `eth.txb.rate-24` and `sum.xb.rate-24` refer to the rates of transmitted bytes over Ethernet interfaces, and total number of received and transmitted bytes, respectively, in node 24 (see Fig. 2).

Non-traffic features include the number of processes, average CPU load, softirq, iowait, process execution time in kernel and user mode, context switches, etc. For the sake of brevity, we do not list here all the features collected in the dataset. The complete list of features with a brief description can be found in the public repository of the dataset [19].

Overall there were 63 nodes gathered in the dataset, with a total of 2387 features. To evaluate the effectiveness of ML methods to detect anomalies we proceeded as follows. On April 14, 2021, one of the two mesh gateways (node 24) failed and was replaced. Due to the node failure, samples from this node were missing between 01:55 and 17:40 of April the 14th. For the testing set, we used the samples gathered during a 3 day interval when the failure occurred: April the 13, 14 and 15th (694 samples). For the training set, we used the samples gathered during the 4 weeks prior to the testing set (7237 samples). Fig. 2 shows the `sum.xb.rate-24` traffic feature of node 24 during the training and testing set.

5. Experiments and results

In this section, we will detail our experiments utilizing unsupervised AD methods. All experiments were run on an IBM Power SC922[1] server with 160 Power9 CPUs clocked at 3.7 GHz, 644 GB of RAM, and 4 Nvidia V100[2] 32 GB GDDR5 GPUs with NVLlink.

[1] https://www.ibm.com/downloads/cas/KQ4BOJ3N
[2] https://www.nvidia.com/en-us/data-center/v100/

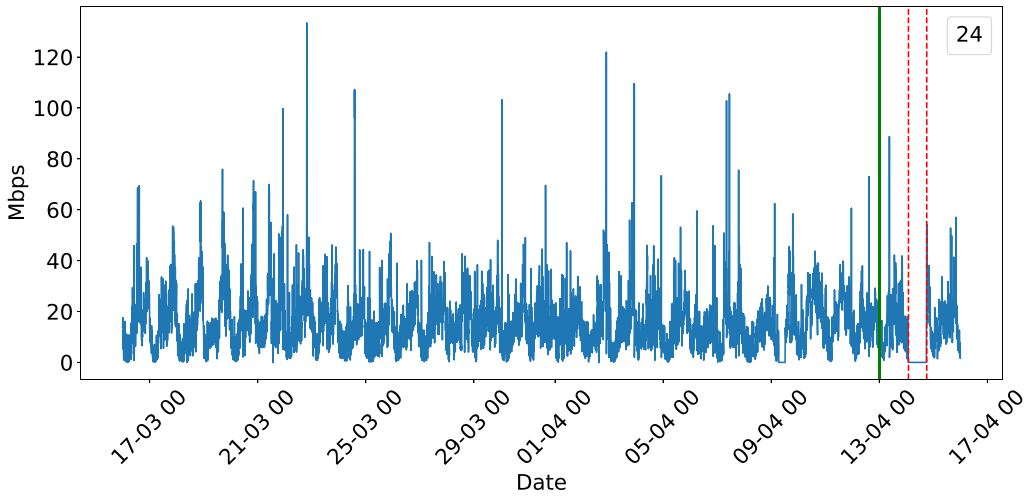


Fig. 2. Plot of the `sum.xb.rate-24` traffic feature (sum of received and transmitted traffic over node 24) during the training and testing set (left and right of the solid lined, respectively). Dashed lines show the gateway failure interval. Peaks and valleys show daily activity and inactivity periods. Dates in the x-axis are formatted as day–month hour.

We have used the 4 ML methods presented in Section 3. These methods were chosen because they are radically different approaches for AD. The experiments were run in python using the libraries scikit-learn [73] for PCA and pyod [74] for Isolation Forest, CBLOF, and VAE. The experiments follow a well-established set of steps. First, the raw data is formatted, cleaned, and normalized. Next, we run several ML based detection methods to find the best performing hyper-parameters. Feature selection methods are then applied to improve these initial results. Finally, we run a second set of experiments with the new subset of the original data and analyze the final outcome. The python scripts used to process the datasets can be found in the public repository https://github.com/llorenc/ml-comcom.

As with all ML tasks, data understanding and cleaning is one of the most time-consuming and crucial steps. In our experiments, we prune those features which had a low variance in the selected time frame. Low variance features have been identified globally, encompassing all nodes from the mesh network. Training data is comprised of 7237 samples each with 1585 features after pruning, while the testing set is 694 samples with the same feature space as the training set.

Data Normalization can seriously degrade the performance of ML methods when incorrectly applied to imbalanced datasets. It is of the utmost importance to normalize discrepancies within the dataset features. Insignificant as well as dominant values should be within an acceptable range [75]. We have chosen the *min–max* technique which normalizes the data in the $[0, 1]$ range based on the following:

$$X_n = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{8}$$

The resulting normalized values denoted by X_n have the benefit of preserving the relationships between features thus virtually eliminating bias. Furthermore, in the case of some ML methods (ex. Deep learning and Neural Network based solutions) the correct normalization of the data reduces the convergence rate. For PCA we have found it more convenient to apply the min–max normalization only to non-traffic features. Traffic features have been normalized by dividing each feature vector by the mean over the maximum of each traffic feature set. PCA is very sensitive to the normalized feature ranges. Thus, doing this way we give more importance to traffic over non-traffic features, while maintaining the relative importance to nodes having higher traffic values.

Most libraries for unsupervised anomaly and outlier detection methods require an expected contamination factor to be defined for training. As anomalous events happen rarely and we wish to avoid false positives as much as possible we set the contamination to a low value of $\alpha = 0.005$. Then, the threshold for a score to be considered anomalous is given by the $1 - \alpha$ quantile of the training set. For our training set of 7237 samples this contamination factor yields $\lceil \alpha \times 7237 \rceil = 37$ anomalies. Note that over the 694 samples of testing set, under normal conditions we would foresee having $\lceil \alpha \times 694 \rceil = 4$ anomalies. However, due to the gateway failure that occurs during the testing interval, we expect the different ML methods to detect a larger number of anomalies.

Table 1
Anomaly detection method hyper-parameters.

ML method	Parameters	Value
PCA	*PC number*	45
Isolation Forest	*n_estimators*	20
	max_samples	0.7
	max_features	1.0
CBLOF	*n_clusters*	8
	alpha	0.9
	beta	5
VAE	*encoder_neurons*	[128, 64, 32]
	decoder_neurons	[32, 64, 128]
	epochs	30
	dropout_rate	0.2
	activation	ReLU

For each of these algorithms, we had to tune the hyper-parameters manually so that anomalous events are grouped inside the time-frame we know anomalous events occurred. Table 1 shows the hyper-parameters used for each ML detection method. These values were fine-tuned using Random Search with a manually defined hyper-parameter search space. In the case of PCA, the only parameter is the number of principal components, PC. To assess the number of PC we have used the common method of choosing the number of dimensions that preserve 95% of residual variance. In the case of CBLOF, we used the default values for all parameters as these provided the best result. VAE parameters include a dropout rate which can help with model generalization and prevent overfitting. The shape of the encoder and decoder parts of VAE is also symmetrical. An important consideration is that we set the batch size to 32. The hardware on which these experiments were executed can handle a substantially larger batch size, however, to keep it in line with what we could expect on more limited hardware, such as Nvidia Jetson Nano[3] board we used a much smaller value. A similar approach has been taken when setting the number of training epochs. We are aware that both batch size and training epochs are important hyper-parameters, however, our ultimate aim is to see

[3] https://developer.nvidia.com/embedded/jetson-nano-developer-kit

Table 2
Results of ML method based experiments.

ML method	Num. of features	Training [s]	Inference [s]	Testing anomalies
PCA	(all) 1585	1.25	0.14	151/122
	(only traffic) 880	0.88	0.11	13/11
Isolation Forest	(all) 1585	2.33	1.04	74/69
	(only traffic) 880	0.97	0.37	10/4
CBLOF	(all) 1585	18.28	2.71	125/115
	(only traffic) 880	3.42	0.09	3/2
VAE	(all) 1585	58.61	1.09	134/122
	(only traffic) 880	39.36	0.49	23/12

how the above-mentioned ML methods will behave on devices that might be located in an Edge/Fog type scenario. We should note that, because we lack a labeled dataset it is virtually impossible to apply normal hyper-parameter optimization techniques. Instead, we focus on finding those hyper-parameters which result in a predictive model with high stability and repeatability. In our case, we gauge this by looking for those models which consistently detect anomalies within our testing interval. We should note that algorithm hyper-parameters are not selected at random nor in isolation as they are, in many cases, linked with one another. Although, in our experiments we used unguided optimization techniques the hyper-parameter space for each algorithm was defined after significant consideration. For example in case of IF a low estimator count with a maximum number of features and samples would lead to model overfitting. In our case for IF estimator count was defined in the range $[10, 100]$, sample size $[0.3, 1.0]$ and features on $[0.2, 10.0]$. Similarly, in case of VAE the number of epochs was set in the range $[5, 100]$ dropout in the range $[0.1, 0.8]$. A total of 3 activation functions types where tested [‘*relu*’, ‘*elu*’, ‘*parametric_relu*’]. Encoder and decoder topology is symmetrical and usually follows descending and ascending pattern respectively. For our experiments the initial neuron count for the encoder was defined in the range $[32, 512]$.

Arguably Isolation Forest was the most difficult to tune. As described in Section 3.2 Isolation Forest has many steps which are affected by randomness to a substantial degree. This is useful in many instances however, it can yield inconsistent results. If no random seed is used on a badly tuned model each training iteration of the Isolation Forest can yield different results. After hundreds of experimental runs, we decided on a relatively small number of estimators (20), maximum number of features for training each estimator and 0.7 as a sample size in order to help avoid overfitting.

The results of all experiments are contained in Table 2. For each algorithm, we list training and inference times as well as the number of anomalies detected in the testing set, and how many of them fall inside the gateway failure interval (122 samples were gathered during the failure interval). Since the gateway failure is clearly an anomalous condition, the number of samples detected as anomalous by the algorithms belonging to the gateway failure interval can be used as a performance measure. For each algorithm, Table 2 shows the anomalies detected using all features and only traffic features. By comparing the anomalies detected in both cases, we see that considering all features significantly increases the performance of the algorithms. For example, with VAE with all features, 134 anomalies were detected, of which all 122 samples were gathered during the gateway failure interval. Using traffic features alone, only 23 anomalies were detected, of which only 12 were within the gateway failure interval. In terms of runtime, we can clearly see that model inference requires less runtime than training. In our case, we only measured inference on the training dataset, since it is an order of magnitude larger than the testing set.

The next step in our experiments was to test the ML methods by varying the size of the dataset. For this purpose, we first varied the number of samples used for the training set. The results are given in Fig. 3, showing the number of anomalies with respect to the training samples gathered during the number of days indicated on the abscissa axis. The training samples are taken from the day immediately prior to the testing set, up to all days of the previous 4 weeks. Fig. 3 shows quite different performance in the different ML methods. While PCA, CBLOF, and VAE generate a large number of anomalies, and thus false positives, when only the samples from a small number of days are considered, Isolation Forest only detects a small number of anomalies and thus has a large number of false negatives. On the other hand, when using the samples of more than 13 days, the anomalies detected by VAE remain almost the same, correctly detecting as anomalous all samples within the gateway failure interval. The number of anomalies detected by the other methods oscillates even after using the samples of more the 20 days for the training set. We can conclude that in our experiments VAE is able to reliably detect anomalies with a smaller number of samples.

Next, we investigated the ML methods by varying the number of features. We chose from one to all features, ordered by significance, and computed the number of anomalies. To compute the significance of the feature we used the contribution plot of the PCA Q statistics (see Section 3.1), and the selection methods detailed in Section 3.5. To compute the Shapely values of the predictions we used the SHAP library [76] which uses several approximation methods to compute the Shapely values. Isolation Forest was chosen to compute the Shapely values of the testing set predictions as it is well supported by the SHAP library and is relatively fast when it comes to training and inference times. As payout (prediction function) we used 0 and 1 for samples classified as normal and anomalous, respectively. Fig. 4 shows the number of anomalies detected in the testing set, and how many of them fall within the gateway failure interval, varying the number of features used for training. The anomalies computed with the ML methods listed in each row are divided according to the type of feature selection method, shown at the top of each column. The number of features is ordered by a decreasing absolute value given by each feature selection method. The figure shows that the studied ML methods behave very differently depending on the number of features used. Isolation Forest is the most sensitive, and even when a large number of features is used, adding just one additional feature can cause the method to significantly change the number of points considered as anomalous. On the other hand, VAE is shown to be the most stable. In fact, using PCA, MAD, and SHAP as the feature selection method, the number of anomalies detected using VAE remains almost constant when the number of features is larger than about 1/4th of the feature set. As for the feature selection methods, Fig. 4 shows that kurtosis gives the worst performance. For the other methods Fig. 4 shows that there is no clear winner and that, depending on the ML algorithm, a different feature selection method may give better results.

As explained above, the contamination parameter set during training produced 37 anomalies out of the 7237 samples. We should also mention that we specifically selected a time frame where we assume no anomalous events have occurred. Even so, except for PCA, the ML methods largely detect the same anomalies. This can be easily seen in Fig. 5.(a). This figure shows the detected overlapping anomalies among the 4 ML methods in the training set. The largest difference occurs between PCA and the other ML methods. This is possibly due to the different normalization applied to PCA. However, if we consider the testing set there is a high agreement among all ML methods, as shown in Fig. 5.(b). First, note that the number of anomalies is much higher than the 4 anomalies that we would expect. Of course, this is due to the gateway failure interval, where most of the anomalies are found, as shown in Fig. 5.(c). For example, Fig. 5.(c) shows that PCA and VAE detect all 122 points in the gateway failure interval as anomalous. However, PCA detects 151 anomalous points in the testing set, while VAE detects only 134. Possibly, the higher number of points detected by PCA is due to overfitting.

To have some insight into the ML methods being compared, Fig. 6 shows a 2D PCA projection of the training set (principal components PC1, PC2) for Isolation Forest, CBLOF, and VAE. Note that the projected

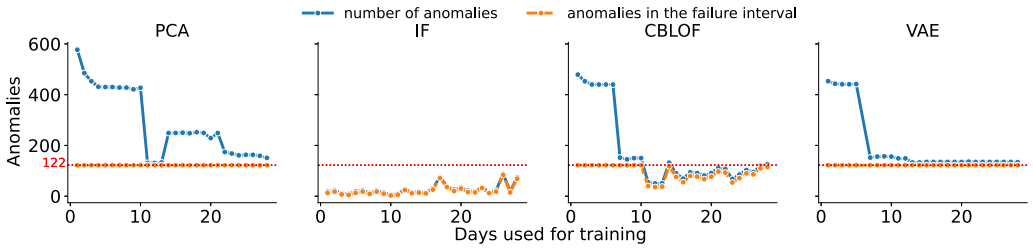


Fig. 3. Number of anomalies found in the testing set varying the number of samples used for training (samples gathered during the previous days). The figure also shows the number of anomalies obtained within the gateway failure interval (dashed line), and the number of points in this interval (122).

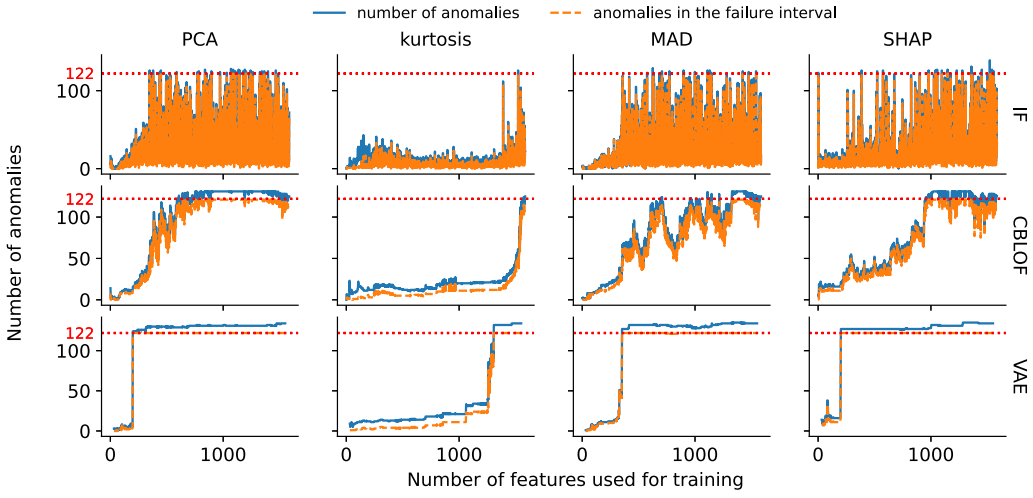


Fig. 4. Number of anomalies found in the testing set, as in Fig. 3, but varying the number of features used for training. The features are selected with different methods (shown at the top of each column).

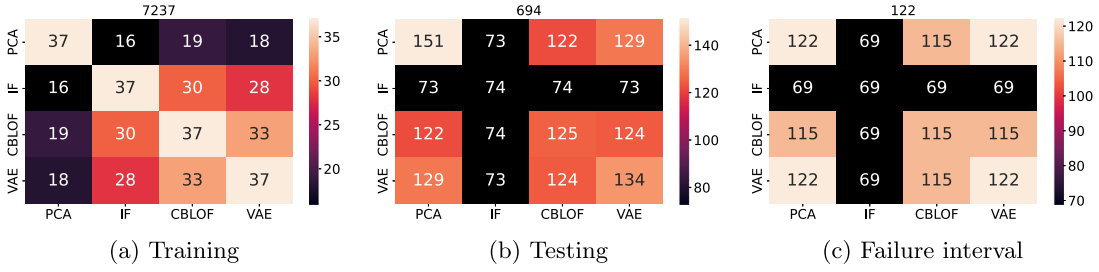


Fig. 5. Detected overlapping anomalies. Each row/column represents an AD method. Each cell of the heatmap represents the number of overlapping anomalies detected by that pair of methods. At the top of each figure there is the total number of points of the set.

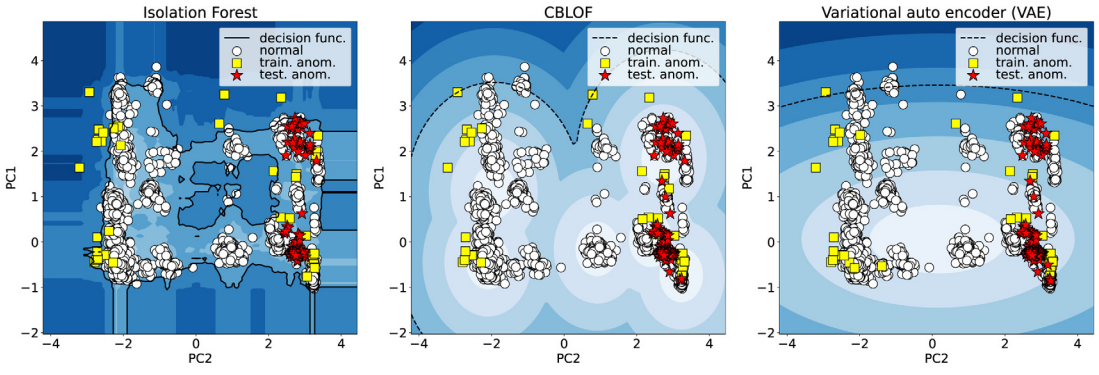
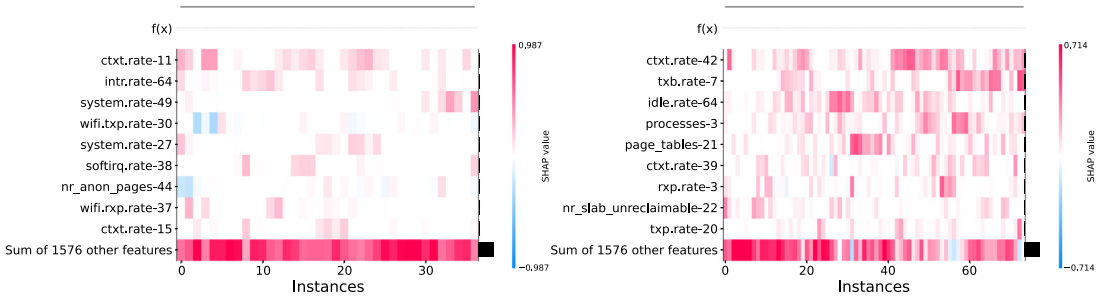


Fig. 6. Decision boundaries of the ML methods re-projected using PCA with 2 components. The projected points are those in the training set (circles), anomalies due to the contamination factor in the training (squares), and anomalies in the test set (stars).



(a) Training (b) Testing

Fig. 7. SHAP heatmap plots for the training and testing anomalies obtained with Isolation Forest. The y-axis shows the model inputs sorted in descending order from top to bottom, for each anomalous event.

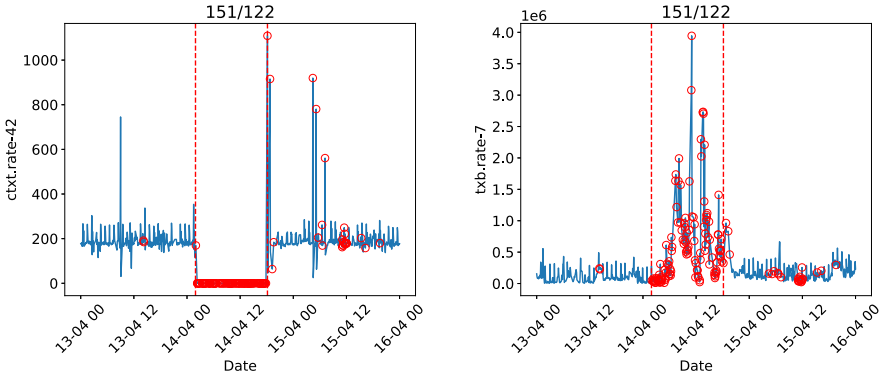


Fig. 8. Features contributing most to SHAP score during the testing set. Vertical lines correspond to the failure interval. Marked samples correspond to anomalies detected by PCA.

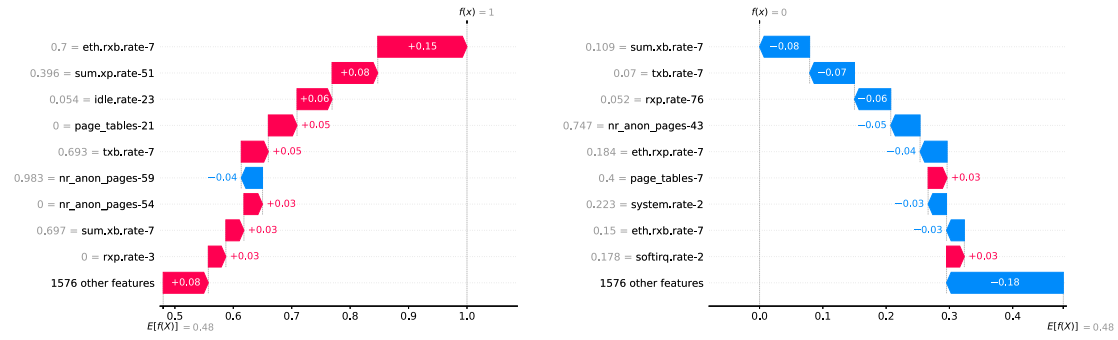
points have the same coordinates since we have used the same scaling for these methods. We have marked the 37 anomalies computed for the training set, and we have also projected the anomalies computed for the testing set. In addition, we have run the methods for the projected training set in 2D and plotted the resulting decision boundary. Note that the anomalies that would be detected using the 2D training set (those points outside the decision boundary) would be quite different from the anomalies detected using the full feature set. We can easily see the differences between how each ML method "decides" what is an anomalous event and what is not. Of particular interest is the decision boundary calculated for Isolation Forest. Since this method is based on the same basic concepts as supervised decision tree models (in particular, Random Forest), we can safely assume that some of the idiosyncrasies can also be found in Isolation Forest. Fig. 6 clearly suggests that model overfitting can be a problem in the case of Isolation Forest. This can lead to poor out-of-sample performance.

For the interpretation of the anomalies, Fig. 7 shows a SHAP heatmap of the anomalies obtained using the training and testing sets, in sub-figures (a) and (b), respectively. On the abscissa axis we see the anomalous events, while the ordinate axis shows the model inputs sorted in decreasing order from top to bottom. The computed Shapely values are on a color scale based on hierarchical clustering and their explanation similarity. At the top of each subfigure is the output of the $f(x)$ function with the predicted values, which is always 1 because we are plotting only the anomalous samples. In Fig. 7 it can be seen that the intensity of the colors of the anomalies obtained for the training set are much lighter and sparser, while they are more intense and frequent for the testing set. This makes it clear that there is a group of features that are significantly affected by the gateway failure during the testing set and, therefore, are assigned a high Shapley value for most of the anomalies detected in the testing set. It is also interesting to note in Fig. 7 that none of the features that contribute most to the anomalies found using the training set (obtained by setting the contamination factor) match those features that contribute most using the testing set (caused by the gateway failure). This fact suggests, as expected, that the anomalies detected in both sets are of different nature.

Fig. 8 shows a time series of the two features that contribute most to the AD in the testing set (as shown in Fig. 7.(b)). These are the CPU context switches rate of node 42 (`ctxt.rate-42`), and the rate of transmitted bytes of node 7 (`txb.rate-7`). Note that `ctxt.rate-42` has an average value around 200 during normal operation, and 0 during the gateway failure. This is because during the gateway failure node 42 was disconnected from the network, and no samples were gathered. These missing samples were set to 0. On the other hand, Fig. 8 shows that `txb.rate-7` has an atypical higher value during the gateway failure. This is because node 7 is close to the gateway that failed, and during the failure it absorbed most of the traffic that was redirected to the other gateway in the mesh.

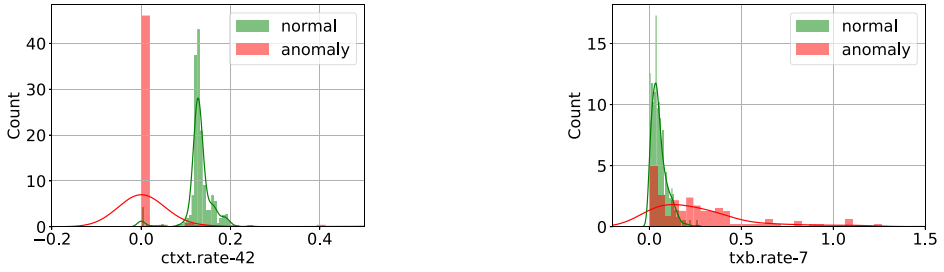
5.1. Discussion

Our experiment results have been interesting, we found that all 4 selected ML detection methods perform well on the given dataset. Unfortunately, in the case of feature selection methods, this is not the case. The method performing best in most cases is based on the calculation of Shapely values which are computationally expensive. This is especially true in the case of large feature spaces as the total number of possible coalitions is 2^n where n is the number of features. The SHAP library we used in our experiments utilizes a permutation based Shapely value estimator. VAE performed the best with the full feature space while Isolation Forest improved the most after Shapely value based feature selection. Overall CBLOF performance was the most consistent with relatively fast training and inference times. The MAD selection method also provided a noteworthy reduction in feature space, especially for Isolation Forest and VAE methods. However, in the case of CBLOF the results are not as stable, recursively adding features



(a) Anomaly (b) Normal

Fig. 9. SHAP representation of an anomalous and normal samples randomly taken from the testing set. At the bottom is the expected value of the model. Each line shows the positive (red) or negative (blue) contribution of each feature to the prediction. The features are sorted in descending order of contribution.



(a) Feature ctxt.rate-42 (b) Feature txb.rate-7

Fig. 10. Feature separability for most impactful features. The figure shows the histograms of the scaled most impactful features for the normal and anomalous events of the testing set.

based on their MAD score can cause a significant dip in predictive performance.

One issue which has come up during our experiments is that we do not know how many anomalous events types there are in the dataset. All 4 methods label samples as anomalous or normal but they do not have a distinction regarding anomaly types. Fig. 9 shows waterfall plots from SHAP representing an anomalous event and a normal event, picket at random from the testing set, features are sorted in decreasing order. We can see that the predictions are $f(x) = 1$ and $f(x) = 0$ for the anomaly, and normal sample, respectively with a base value of $E[f(x)] = 0.48$. This base value represents the average of the target variable for all events from the dataset. The features colored in red, shift the predictions towards 1 (anomalous event) while those marked in blue shift them towards 0 (normal event). The separation of predicted events can be easily observed. This also remains true when we plot histograms for the most impactful features of the testing set, as seen in Fig. 10.

This information although useful, will still not answer the question: How many anomaly types are there and how does the reduction in feature space affect prediction? It is highly likely that by reducing the feature space we effectively remove the capacity of our detection methods from detecting any other anomaly type. For example, if we only have CPU metrics in a dataset any memory-related anomalies would be almost impossible to find.

To answer this question we decided to cluster the detected anomalies using a method that does not force each datapoint (event) to belong to a cluster and also has the concept of noise. For this we selected the Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDDBSCAN) [77] algorithm which in itself is also a method used for AD. It performs DBSCAN [78] over varying epsilon (*eps*) values and integrates the result to find a clustering of the data points with the best stability over *eps*.

In standard DBSCAN *eps* specifies how close points should be to each other to be considered part of a cluster. The setting of this parameter is

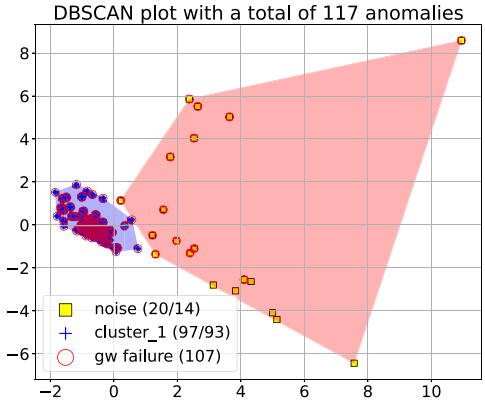


Fig. 11. Anomalous event clustering using HDDBSCAN. On the left is the anomalous event cluster (97 points), and on the right is the noise (20 points). 107 points correspond to gateway failure (circles), of which 93 are in the anomalous cluster, and 14 in the noise.

a major challenge, if it is set too small a substantial part of the data will not be assigned to a cluster and marked as outliers. If it is too high all clusters are merged into a single large cluster. HDDBSCAN eliminates this issue by returning meaningful clusters with little parameter tuning necessary.

In our case, we set the minimum cluster size to 30. For consistency, we used the anomalies identified by Isolation Forest. Fig. 11 shows that HDDBSCAN identified one cluster containing 97 anomalous events and 20 anomalous events identified as noise. From these anomalies, 107 correspond to the gateway failure interval, 14 of them being classified as noise, and 93 in the cluster. Our interpretation of these results is that most anomalous events we have identified are of the same type, which corresponds to the gateway failure.

When it comes to the complexity of the selected ML methods we should first state that IF is an algorithm especially suited for processing high volumes of data having a linear time complexity with a low constant and low potential for memory requirements (depending on the hyper-parameters used) [1]. Similarly, CBLOF is also well suited for large volumes of data $O(N \log N)$ in the best case scenario [55]. Lastly, VAE is notorious for its high computational complexity which is largely based on the network topology being used during training. However, in our experiments this added complexity resulted in interesting results. VAE being able to outperform other methods while having a minimal topology. A topology which is a direct result of the structure of the dataset used. Also, VAE models was the most consistent during hyper-parameter optimization, resulting in workable models in all but the most extreme hyper-parameter values.

We should also point out that our experiments encompass several additional ML methods. The results presented in this paper are based on the performance of each model. For example, we have performed experiments using GAN-based methods such as ALAD [79] and AnoGAN [80]. The main difference between the two is that ALAD is built on top of a two-way GAN which avoids the computationally expensive inference of AnoGAN. Unfortunately experimental results have been poor in both cases. ALAD had erratic results, even with a fixed random seed and hyper-parameters, the resulting models gave different results, while the AnoGAN training took a long time (well over 2 hours) with limited results. GAN-based networks are known to be difficult to train. They suffer from mode collapse and generate only a subset of the original dataset [62].

Similarly, we ran several experiments using $\beta - VAE$ with the values for β ranging from $[0.1 < 10.0]$. However, the best results where obtained using the default β value of 1.0 included in the original paper. This was not completely unexpected as there are some issues with reproducibility of $\beta - VAE$ results [81]. Finally, we also used an implementation of Deep-SVDD [82] in our experiments with results notably worse than the algorithms presented in Section 4.

For brevity, we did not include all the algorithms in our experiments, we selected 3 that have the best performance and also have different underlying principles. However, the code and hyper-parameter settings used for these experiments are available in our repository.

We conclude that the lack of predictive performance of some of the methods used in our experiments is due to the large feature space of our dataset. Although we find that the use of feature selection techniques performs well for the three best performing models (see Fig. 3) future work will focus on additional methods. Non-negative matrix factorization (NMF) [83] is promising, but our initial experiments with it have produced limited results. Future work will focus on addressing this issue and include a reformulation of the training and testing datasets including graphs of the routing tables used in the mesh network as well as the inclusion of additional AD methods.

6. Conclusion

In the last decade, machine learning, ML, methods have exploded and have been applied to an increasing number of fields. Computer networks are no exception, and numerous works have used ML to investigate network intrusion detection. Fault detection in computer networks is an attractive application for ML that has received little attention, largely due to the lack of available datasets. In this paper, we attempt to fill this gap by performing anomaly detection, for fault detection using ML. The MLs chosen for our study are based on 4 very different principles. The first one is Principal Component Analysis. PCA is a well-known method that has been widely used in the manufacturing industry. PCA is based on the projection onto a reduced dimension subspace that preserves maximum variance. The second is Isolation Forest, IF. IF is based on the idea of constructing decision trees that isolate the samples into different classes, and take as anomalous those that are isolated in a smaller number of steps. The third one, Clustering-Based Local Outlier Factor, CBLOF, uses the clustering principle, choosing as anomalous those samples that have larger distances to the formed clusters. Finally, we have used Variational AutoEncoders, VAE, which is based on a deep neural network model. All these methods have been shown in the literature to be particularly suitable for AD.

Unlike other works found in the literature, we have constructed a dataset of a production network. For our dataset we used an interval in which an unprovoked gateway failure occurred. This rare event allowed us to analyze the performance of the ML methods under study by checking how many of the samples gathered during the gateway failure interval are flagged as anomalous. The dataset was constructed by collecting samples every 5 minutes. The data samples have not only traffic characteristics, but also non-traffic characteristics, such as CPU usage, memory usage, etc. We used a period of 4 weeks for the training set, and a period of 3 days for the testing set. The gateway failure occurs for 16 hours in the middle of the testing set. The main conclusions can be summarized as follows:

- When properly tuned, the anomalies derived from gateway failure are well captured by all ML methods.
- Outside the failure interval several traffic spikes are also marked as anomalies. These spikes, however, can be considered as noise, and a consequence of irregular traffic patterns of users traffic.
- The deep learning method based on VAE outperforms the other ML methods, at cost of significantly higher computational cost.
- Considering other features related to CPU and memory in addition to traffic features significantly increases the number of points detected as anomalous inside the gateway failure interval.
- Of the feature selection methods we tested, mean absolute difference, MAD, and Shapely value, SHAP, performed the best. MAD is much easier to calculate, while SHAP provides a better understanding of why events are flagged as anomalous (explainability).

CRediT authorship contribution statement

Llorenç Cerdà-Alabern: Conceptualization, Investigation, Software, Writing, Dataset gathering. **Gabriel Iuhasz:** Conceptualization, Investigation, Software, Writing. **Gabriele Gemmi:** Investigation, Writing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The dataset is available at the link to Zenodo provided in the paper.

Acknowledgments

This work has received funding through the DiPET CHIST-ERA under grant agreement PCI2019-111850-2; Spanish grant PID2019-106774RB-C21; Romanian DIPET (62652/15.11.2019) project funded via PN 124/2020; and has been partially supported by the EU research project SERRANO (101017168) and hardware resources courtesy of the Romanian Ministry of Research and Innovation UEFISCDI COCO research project PN III-P4-ID-PCE-2020-0407.

References

[1] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM Comput. Surv. 41 (3) (2009) 1–58.

[2] C.C. Aggarwal, Outlier Analysis, Springer, 2017.

[3] M. Ahmed, A.N. Mahmood, J. Hu, A survey of network anomaly detection techniques, J. Netw. Comput. Appl. (60) (2016) 19–31.

[4] D.P. Kumar, T. Amgoth, C.S.R. Annavarapu, Machine learning algorithms for wireless sensor networks: A survey, Inf. Fusion 49 (2019) 1–25.

[5] M. Mohri, A. Rostamizadeh, A. Talwalkar, Foundations of Machine Learning, MIT Press, 2018.

[6] K.P. Murphy, Probabilistic Machine Learning: An Introduction, MIT Press, 2022.

[7] L.N. Tidjon, M. Frappier, A. Mammar, Intrusion detection systems: A cross-domain overview, IEEE Commun. Surv. Tutor. 21 (4) (2019) 3639–3681.

[8] G. Fernandes, J.J. Rodrigues, L.F. Carvalho, J.F. Al-Muhtadi, M.L. Proença, A comprehensive survey on network anomaly detection, Telecommun. Syst. 70 (3) (2019) 447–489.

[9] D. Vega, R. Baig, L. Cerdà-Alabern, E. Medina, R. Meseguer, L. Navarro, A technological overview of the guifi.net community network, Comput. Netw. 9 (2) (2015) 260–278, http://dx.doi.org/10.1016/j.comnet.2015.09.023.

[10] Y. Ben David, Connecting the Last Billion (Ph.D. thesis), UC Berkeley, 2015.

[11] Guifi.net, Open, free and neutral network internet for everybody, 2021, http://guifi.net/en. (Accessed 13 January 2021).

[12] L. Cerdà-Alabern, R. Baig, L. Navarro, On the guifi.net community network economics, Comput. Netw. 168 (2020) 107067.

[13] GuifiSants, Xarxa oberta, lliure i neutral del barri de sants, 2021, http://sants.guifi.net/. (Accessed January 2021).

[14] J. Camacho, A. Pérez-Villegas, P. Garcí a Teodoro, G. Maciá-Fernández, PCA-based multivariate statistical network monitoring for anomaly detection, Comput. Secur. 59 (2016) 118–137.

[15] D.H. Hoang, H.D. Nguyen, A PCA-based method for IoT network traffic anomaly detection, in: 2018 20th International Conference on Advanced Communication Technology, ICACT, IEEE, 2018, pp. 381–386.

[16] I.K. Savvas, A.V. Chernov, M.A. Butakova, C. Chaikalis, Increasing the quality and performance of N-dimensional point anomaly detection in traffic using PCA and DBSCAN, in: 2018 26th Telecommunications Forum, TELFOR, IEEE, 2018, pp. 1–4.

[17] Munin networked resource monitoring tool, http://munin-monitoring.org.

[18] Nagios, The Industry Standard In IT Infrastructure Monitoring, https://www.nagios.org.

[19] L. Cerdà-Alabern, Dataset for Anomaly Detection in a Production Wireless Mesh Community Network, Zenodo, 2022, http://dx.doi.org/10.5281/zenodo.6169917.

[20] V. Hodge, J. Austin, A survey of outlier detection methodologies, Artif. Intell. Rev. 22 (2) (2004) 85–126.

[21] S. Northcutt, J. Novak, Network Intrusion Detection, Sams Publishing, 2002.

[22] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E.D. Kolaczyk, N. Taft, Structural analysis of network traffic flows, in: Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, 2004 pp. 61–72.

[23] Z.R. Zaidi, S. Hakami, T. Moors, B. Landfeldt, Detection and identification of anomalies in wireless mesh networks using principal component analysis (PCA), J. Interconnect. Netw. 10 (04) (2009) 517–534.

[24] Z.R. Zaidi, S. Hakami, B. Landfeldt, T. Moors, Real-time detection of traffic anomalies in wireless mesh networks, Wirel. Netw. 16 (6) (2010) 1675–1689.

[25] C. Pascoal, M.R. De Oliveira, R. Valadas, P. Filzmoser, P. Salvador, A. Pacheco, Robust feature selection and robust PCA for internet traffic anomaly detection, in: 2012 Proceedings IEEE Infocom, IEEE, 2012, pp. 1755–1763.

[26] H. Ringberg, A. Soule, J. Rexford, C. Diot, Sensitivity of PCA for traffic anomaly detection, in: Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, 2007 pp. 109–120.

[27] N.L.D. Khoa, T. Babaie, S. Chawla, Z. Zaidi, Network anomaly detection using a commute distance based approach, in: 2010 IEEE International Conference on Data Mining Workshops, IEEE, 2010, pp. 943–950.

[28] A.B. Nassif, M.A. Talib, Q. Nasir, F.M. Dakalbab, Machine learning for anomaly detection: A systematic review, IEEE Access 9 (2021) 78658–78700, http://dx.doi.org/10.1109/ACCESS.2021.3083060.

[29] S. Goldstein, A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data, PLoS One 11 (4) (2016) 1–31, http://dx.doi.org/10.1371/journal.pone.0152173.

[30] M. Ahmed, A.N. Mahmood, Novel approach for network traffic pattern analysis using clustering-based collective anomaly detection, Ann. Data Sci. 2 (2015) 111–130.

[31] X. Chun-Hui, S. Chen, B. Cong-Xiao, L. Xing, Anomaly detection in network management system based on isolation forest, in: 2018 4th Annual International Conference on Network and Information Systems for Computers, ICNISC, 2018, pp. 56–60, http://dx.doi.org/10.1109/ICNISC.2018.00019.

[32] G. Pang, C. Shen, L. Cao, A.V.D. Hengel, Deep learning for anomaly detection: A review, ACM Comput. Surv. 54 (2) (2021) http://dx.doi.org/10.1145/3439950.

[33] N. Takeishi, Y. Kawahara, On anomaly interpretation via Shapley values, 2020, arXiv:2004.04464.

[34] L. Antwarg, R.M. Miller, B. Shapira, L. Rokach, Explaining anomalies detected by autoencoders using Shapley additive explanations, Expert Syst. Appl. 186 (2021) 115736, http://dx.doi.org/10.1016/j.eswa.2021.115736.

[35] C. Zhou, R.C. Paffenroth, Anomaly detection with robust deep autoencoders, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 665–674, http://dx.doi.org/10.1145/3097983.3098052.

[36] D.P. Kingma, M. Welling, Auto-encoding variational Bayes, 2014, arXiv:1312.6114.

[37] M. Moulay, R.G. Leiva, V. Mancuso, P.J.R. Maroni, A.F. Anta, Ttrees: Automated classification of causes of network anomalies with little data, in: 2021 IEEE 22nd International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM, IEEE, 2021, pp. 199–208.

[38] K. Sequeira, M. Zaki, ADMIT: Anomaly-based data mining for intrusions, in: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '02, Association for Computing Machinery, New York, NY, USA, 2002, pp. 386–395, http://dx.doi.org/10.1145/775047.775103.

[39] Y.F. Zhang, Z.Y. Xiong, X.Q. Wang, Distributed intrusion detection based on clustering, in: 2005 International Conference on Machine Learning and Cybernetics, vol. 4, 2005, pp. 2379–2383 Vol. 4, http://dx.doi.org/10.1109/ICMLC.2005.1527342.

[40] M.H. Bhuyan, D.K. Bhattacharyya, J.K. Kalita, An effective unsupervised network anomaly detection method, in: Proceedings of the International Conference on Advances in Computing, Communications and Informatics, ICACCI '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 533–539, http://dx.doi.org/10.1145/2345396.2345484.

[41] N. Hu, Z. Tian, H. Lu, X. Du, M. Guizani, A multiple-kernel clustering based intrusion detection scheme for 5G and IoT networks, Int. J. Mach. Learn. Cybern. 12 (2021) http://dx.doi.org/10.1007/s13042-020-01253-w.

[42] M.E. Otey, A. Ghoting, S. Parthasarathy, Fast distributed outlier detection in mixed-attribute data sets, Data Min. Knowl. Discov. 12 (2–3) (2006) 203–228, http://dx.doi.org/10.1007/s10618-005-0014-6.

[43] M. Bhuyan, D.K. Bhattacharyya, J. Kalita, A multi-step outlier-based anomaly detection approach to network-wide traffic, Inform. Sci. 348 (2016) http://dx.doi.org/10.1016/j.ins.2016.02.023.

[44] P. Casas, J. Mazel, P. Owezarski, Unsupervised network intrusion detection systems: Detecting the unknown without knowledge, Comput. Commun. 35 (7) (2012) 772–783, http://dx.doi.org/10.1016/j.comcom.2012.01.016.

[45] O. Iraqi, H. El Bakkali, Application-level unsupervised outlier-based intrusion detection and prevention, Secur. Commun. Netw. 2019 (2019) 1–13, http://dx.doi.org/10.1155/2019/8368473.

[46] M. Khan, Rule based network intrusion detection using genetic algorithm, Int. J. Comput. Appl. 18 (2011) 26–29, http://dx.doi.org/10.5120/2303-2914.

[47] H. Alsaadi, R. Almuttairi, O. Ucan, O. Bayat, An adapting soft computing model for intrusion detection system, Comput. Intell. 01 (2021) http://dx.doi.org/10.1111/coin.12433.

[48] A. Shenfield, D. Day, A. Ayesh, Intelligent intrusion detection systems using artificial neural networks, ICT Express 4 (2) (2018) 95–99, http://dx.doi.org/10.1016/j.icte.2018.04.003, SI on Artificial Intelligence and Machine Learning.

[49] C.F. Alcala, S.J. Qin, Analysis and generalization of fault diagnosis methods for process monitoring, J. Process Control 21 (3) (2011) 322–330.

[50] P. Miller, R.E. Swanson, C.E. Heckler, Contribution plots: A missing link in multivariate quality control, Appl. Math. Comput. Sci. 8 (4) (1998) 775–792.

[51] C.C. Aggarwal, S. Sathe, Theoretical foundations and algorithms for outlier ensembles, SIGKDD Explor. Newsl. 17 (1) (2015) 24–47, http://dx.doi.org/10.1145/2830544.2830549.

[52] C.C. Aggarwal, Outlier ensembles: Position paper, SIGKDD Explor. Newsl. 14 (2) (2013) 49–58, http://dx.doi.org/10.1145/2481244.2481252.

[53] F.T. Liu, K.M. Ting, Z.H. Zhou, Isolation-based anomaly detection, ACM Trans. Knowl. Discov. Data 6 (1) (2012) http://dx.doi.org/10.1145/2133360.2133363.

[54] F.T. Liu, K.M. Ting, Z.H. Zhou, Isolation forest, in: 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 413–422, http://dx.doi.org/10.1109/ICDM.2008.17.

[55] Z. He, X. Xu, S. Deng, Discovering cluster-based local outliers, Pattern Recognit. Lett. 24 (9) (2003) 1641–1650, http://dx.doi.org/10.1016/S0167-8655(03)00003-5.

[56] M.M. Breunig, H.P. Kriegel, R.T. Ng, J. Sander, LOF: Identifying density-based local outliers, SIGMOD Rec. 29 (2) (2000) 93–104, http://dx.doi.org/10.1145/335191.335388.

[57] R. Zheng, J. Gu, Anomaly detection for power system forecasting under data corruption based on variational auto-encoder, in: 8th Renewable Power Generation Conference, RPG 2019, 2019, pp. 1–6, http://dx.doi.org/10.1049/cp.2019.0461.

[58] R. Yao, C. Liu, L. Zhang, P. Peng, Unsupervised anomaly detection using variational auto-encoder based feature extraction, in: 2019 IEEE International Conference on Prognostics and Health Management, ICPHM, 2019, pp. 1–7, http://dx.doi.org/10.1109/ICPHM.2019.8819434.

[59] Y. Aizenbud, O. Lindenbaum, Y. Kluger, Probabilistic robust autoencoders for anomaly detection, 2021, arXiv:2110.00494.

[60] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (56) (2014) 1929–1958, http://jmlr.org/papers/v15/srivastava14a.html.
[61] C.P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, A. Lerchner, Understanding disentangling in β-VAE, CoRR (2018) arXiv:1804.03599.
[62] L. Zhou, W. Deng, X. Wu, Unsupervised anomaly localization using VAE and beta-VAE, 2020, http://dx.doi.org/10.48550/ARXIV.2005.10686.
[63] J. Cai, J. Luo, S. Wang, S. Yang, Feature selection in machine learning: A new perspective, Neurocomputing 300 (2018) 70–79, http://dx.doi.org/10.1016/j.neucom.2017.11.077.
[64] C. Suman, S. Tripathy, S. Saha, Building an effective intrusion detection system using unsupervised feature selection in multi-objective optimization framework, 2019, arXiv preprint arXiv:1905.06562.
[65] A.J. Ferreira, M.A.T. Figueiredo, Efficient feature selection filters for high-dimensional data, Pattern Recognit. Lett. 33 (13) (2012) 1794–1804, http://dx.doi.org/10.1016/j.patrec.2012.05.019.
[66] L.S. Shapley, 17. A value for n-person games, in: H.W. Kuhn, A.W. Tucker (Eds.), Contributions to the Theory of Games (AM-28), vol. II, Princeton University Press, 2016, pp. 307–318, http://dx.doi.org/10.1515/9781400881970-018.
[67] C. Molnar, Interpretable Machine Learning, Independently published, 2022.
[68] OpenWrt Project, OpenWrt project: Welcome to the OpenWrt project, 2021, https://openwrt.org/. (Accessed January 2021).
[69] BMX6 mesh networking protocol, http://bmx6.net. (Accessed January 2021).
[70] L. Cerdà-Alabern, A. Neumann, L. Maccari, Experimental evaluation of BMX6 routing metrics in a 802.11an wireless-community mesh network, in: 2015 3rd International Conference on Future Internet of Things and Cloud, 2015 pp. 770–775.
[71] GuifiSants, qMp Sants-UPC, 2021, http://dsg.ac.upc.edu/qmpsu. (Accessed January 2021).
[72] L. Cerdà-Alabern, A. Neumann, P. Escrich, Experimental evaluation of a wireless community mesh network, in: The 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM'13, ACM, Barcelona, Spain, 2013.
[73] F. Pedregosa, et al., Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[74] Y. Zhao, Z. Nasrullah, Z. Li, Pyod: A Python toolbox for scalable outlier detection, J. Mach. Learn. Res. 20 (96) (2019) 1–7, http://jmlr.org/papers/v20/19-011.html.
[75] A. Mahfouz, A. Abuhussein, D. Venugopal, S. Shiva, Ensemble classifiers for network intrusion detection using a novel network attack dataset, Future Internet 12 (11) (2020) http://dx.doi.org/10.3390/fi12110180.
[76] S.M. Lundberg, S.I. Lee, A unified approach to interpreting model predictions, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, Curran Associates, Inc., 2017, pp. 4765–4774.
[77] L. McInnes, J. Healy, S. Astels, Hdbscan: Hierarchical density based clustering, J. Open Source Softw. 2 (11) (2017) 205, http://dx.doi.org/10.21105/joss.00205.
[78] M. Ester, H.P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD '96, AAAI Press, 1996, pp. 226–231.
[79] H. Zenati, M. Romain, C.S. Foo, B. Lecouat, V.R. Chandrasekhar, Adversarially learned anomaly detection, 2018, http://dx.doi.org/10.48550/ARXIV.1812.02288.
[80] T. Schlegl, P. Seeböck, S.M. Waldstein, U. Schmidt-Erfurth, G. Langs, Unsupervised anomaly detection with generative adversarial networks to guide marker discovery, 2017, http://dx.doi.org/10.48550/ARXIV.1703.05921.
[81] M. Fil, M. Mesinovic, M. Morris, J. Wildberger, Beta-VAE reproducibility: Challenges and extensions, 2021, http://dx.doi.org/10.48550/ARXIV.2112.14278, https://arxiv.org/abs/2112.14278.
[82] P. Liznerski, L. Ruff, R.A. Vandermeulen, B.J. Franks, M. Kloft, K.R. Müller, Explainable deep one-class classification, 2020, http://dx.doi.org/10.48550/ARXIV.2007.01760.
[83] H. Alshammari, O. Ghorbel, M. Aseeri, M. Abid, Non-negative matrix factorization (NMF) for outlier detection in wireless sensor networks, in: 2018 14th International Wireless Communications & Mobile Computing Conference, IWCMC, 2018, pp. 506–511, http://dx.doi.org/10.1109/IWCMC.2018.8450421.