mathematics

Article

Design and Evaluation of Unsupervised Machine Learning Models for Anomaly Detection in Streaming Cybersecurity Logs

Carmen Sánchez-Zas *, Xavier Larriva-Novo, Víctor A. Villagrá, Mario Sanz Rodrigo and José Ignacio Moreno

ETSI Telecomunicación, Universidad Politécnica de Madrid (UPM), Avda. Complutense 30, 28040 Madrid, Spain
* Correspondence: carmen.szas@upm.es

Abstract: Companies, institutions or governments process large amounts of data for the development of their activities. This knowledge usually comes from devices that collect data from various sources. Processing them in real time is essential to ensure the flow of information about the current state of infrastructure, as this knowledge is the basis for management and decision making in the event of an attack or anomalous situations. Therefore, this article exposes three unsupervised machine learning models based on clustering techniques and threshold definitions to detect anomalies from heterogeneous streaming cybersecurity data sources. After evaluation, this paper presents a case of heterogeneous cybersecurity devices, comparing WSSSE, Silhouette and training time metrics for all models, where K-Means was defined as the optimal algorithm for anomaly detection in streaming data processing. The anomaly detection's accuracy achieved is also significantly high. A comparison with other research studies is also performed, against which the proposed method proved its strong points.

Keywords: machine learning; clustering; real-time; data pre-processing; threshold; Spark; cybersecurity; K-means; anomaly detection; logs

MSC: 68T99

check for updates

Citation: Sánchez-Zas, C.; Larriva-Novo, X.; Villagrá, V.A.; Rodrigo, M.S.; Moreno, J.I. Design and Evaluation of Unsupervised Machine Learning Models for Anomaly Detection in Streaming Cybersecurity Logs. *Mathematics* **2022**, *10*, 4043. https://doi.org/10.3390/math10214043

Academic Editor: Denis N. Sidorov

Received: 29 September 2022
Accepted: 24 October 2022
Published: 31 October 2022

1. Introduction

The analysis and subsequent extraction of information from heterogeneous volumes of data processed in business environments are computationally intensive, especially when anomaly detection must be performed immediately and in real time. Therefore, security experts began to use methods and mechanisms from one area of computer science that has experienced exponential growth in recent years: the fields of machine learning and Big Data.

Unsupervised learning models [1] attract the most attention because they are able to generate knowledge about unidentifiable events or behaviours. They are supplied with unlabelled data, so knowledge must be extracted from the metastructure of the supplied information itself. This feature is essential for the analysis of most data generated by devices, as these data are not labelled.

Furthermore, in the field of Big Data, various mechanisms and models have been developed to handle and process these volumes of data in real time. This property is critical in security environments where devices are continuously transmitting data.

Regarding anomaly detection, currently, there is a challenge in dealing with large volumes of information feeds in real time, which come from a wide variety of cybersecurity sources. These heterogeneous data must be processed upon receipt using models trained to detect anomalous traffic via unsupervised algorithms that can identify normal traffic and the incoming input is compared to classify it as normal or abnormal. These models would follow similar training, and each one is focused on attacks from the logs of a cybersecurity device.

The motivation of this research is to develop an unsupervised machine learning model to detect real-time anomalies in a custom environment with heterogeneous log sources that monitor communications or behaviours. The presence of different devices as input data is the triggering point for gathering our own dataset, which was labeled by experts by the analysis of the parameters of the different logs, and led us to using unsupervised detecting techniques.

The aim of this work is therefore to present a real environment based on a scalable open-source system that allows the management of large amounts of data in real time by parallelising the work, to develop an unsupervised learning system that can detect anomalies in a set of data from different devices in real time. The machine learning models are trained with pre-processed logs representing normal traffic from each source. This step is vital to assure a correct input to the system. The anomaly detection architecture is based on the development of a thresholding system in conjunction with the clustering algorithm selected after its comparison, which makes it possible to classify which data from the different data sources are anomalous based on the clusters previously formed by the algorithm. Metrics such as WSSSE and Silhouette are used to optimise the model via its hyperparameters and, after the system is developed, its behaviour is evaluated. With the tests performed, whether the model correctly clasifies data as anomalous/non-anomalous is determined in order to obtain a comparison among the different algorithms applied.

Finally, the devices mentioned above that provide the data sources are integrated into the system, as well as a module where the results can be stored, managed and analysed.

With this proposal, the objective is to process logs from heterogeneous cybersecurity devices that are streaming in real time using similarly trained models to detect whether any communication is anomalous.

To test its functioning, we also developed a use case representing a system working in a real environment. Therefore, the article proposes the use of Apache Spark [2], as mentioned in [3], particularly one of the Python APIs offered by Spark, PySpark [4], which includes a library of machine learning models (MLlib) for processing streaming logs from these heterogeneous sources. Unsupervised learning algorithms can be run and integrated into the overall Spark ecosystem to accomplish the task of detecting anomalies in real time. Similarly, the environment provides all necessary tools for data pre-processing and a subsequent connection to an ELK [5] system (ElasticSearch, Logstash and Kibana) where the result can be visualised and analysed. Three unsupervised learning models are proposed: K-Means, Bisecting K-Means and GMM, as they are among the best known unsupervised learning models that allow grouping data into clusters [6].

Training of the model is performed with anomaly-free datasets generated for each source, which are different physical and logical devices (Wi-Fi, Bluetooth, Mobile Networks, Radio Frequency, UEBA and SIEM and Firewall log sources) that represent various logs existing in business environments. The correct modelling of the system requires pre-processing the data with the PySpark tool mentioned earlier so that the algorithms can be trained correctly.

The results of the research highlight the KMeans model as the optimal model for the case under study, obtaining better metrics and prediction results close to 99%.

For these reasons, the main findings of this research are as follows: the process of obtaining datasets formed from the logs of different devices and synthetic data generation, real-time processing of data using an ensemble of several models trained specifically for each data type to detect anomalies.

During the course of this document, in Section 2, we will introduce a general overview of the state-of-the-art methods by analysing previous projects related to the state of the art. In Section 3, an overview of technologies involved in this development is provided. Finally, in Sections 4–6, the defined proposal, its implementation and the results obtained in the validation are provided, and the system is checked in terms of whether it meets the requirements set out in a real scenario. To sum up, Section 7 will include conclusions, exposing the advantages and disadvantages offered by the architecture as well as possible improvements and future lines that may result from the project.

2. Related Works

To solve the problem of detecting anomalies in real time, many studies have been presented to create a mechanism capable of accomplishing it in an effective, simple and powerful manner.

The approach to this problem means that most of current studies need to use Big Data technologies, such as Apache Spark, because these systems enable them in handling the large amount of data that has to be processed in real time. From this point of view, an attempt on addressing the problem via unsupervised clustering, as there is a lack of tagged data to train a supervised algorithm.

The authors in [7] use a K-Means clustering algorithm to tag an unlabelled dataset in order to use it as a basis for training supervised algorithms. They use service and customer data collected from IoT networks to detect abnormal SIM cards.

The proposal from [8] makes use of principal component analysis (PCA) to reduce dimensionality and to apply its results to a Mini Batch K-Means. This proposal allows improvements in execution time as well as the CH metric that evaluates the formed clusters.

A similar proposal to the one described in this document, in which the use of thresholds in each of the clusters formed to eliminate or detect the outliers, noise or anomalies, is described in [9], which is a framework for analysing large IoT datasets by using the parallelisation of the implementation of K-mean methods and using Spark to, afterwards, apply outlier detection to remove fraudulent data.

The authors in [10] propose a method that combines clustering techniques and Support Vector Machines (SSC-OCSVM) to detect anomalies in the NSL-KDD dataset. The researchers in [11] describe a system to detect anomalies from streaming IoT sensors using statistical and deep-learning-based models.

In [12], the authors present their unsupervised anomaly detection method based on autoencoders with Generative Adversarial Networks using five public datasets (SWaT, WADI, SMD, SMAP and MSL) and an internal dataset. Meanwhile, in [13], they propose a real-time implementation of the isolated forest to detect anomalies in the Aero-Propulsion System Simulation dataset.

In [14], researchers use an Adversarial Auto Encoder (AAE) model in wireless spectrum data anomaly detection, compression and signal classification over three spectrum datasets along with one synthetic anomaly set to test the model in a controlled environment. Moreover, the authors in [15] describe an unsupervised anomaly detection approach based on Bluetooth tracking data and the isolation forest algorithm.

In the Results Section, we will present a summary of the procedures of these related research studies in comparison with our work.

An important aspect that is taken into account in various related studies is the metric upon which the results are evaluated. As there are no labels for comparing and measuring the performance of the model, we apply various metrics that provide an indirect evaluation of it. In this way, studies such as [16] compile different metrics in a clustering algorithm.

Another approach is the application of non-iterative methods, such as geometric data transformations (GTM) [17] and the successive geometric transformations model (SGTM) [18].

It can be seen that the efforts and attempts for achieving an unsupervised anomaly detector are quite numerous. However, despite the number of proposals, a concrete architecture to solve the problem is not yet in sight. Therefore, this paper proposes an unsupervised method for anomaly detection using a paradigm based on training models with normal data and detection based on the misclassification of events relative to the clusters formed. We will also apply different metrics and methods of visualising the clusters to obtain a more robust idea of what occurs during clustering and so that we aim to evaluate the performance of the clusters more comprehensively.

3. Unsupervised Learning

3.1. Clustering

As one of the large families of unsupervised learning, the main task of clustering models [19] is the automatic grouping of unlabelled data to build subsets of data known as clusters. Each cluster is a collection of data that, according to certain metrics, are similar to each other. By extension, data belonging to different clusters have differentiating elements, thus providing a simultaneous dual function: the aggregation of data that have similar elements between them and the segmentation of data that have non-common characteristics. This aggregation/segmentation that occurs when grouping data into clusters, as indicated above, is conditional on the method of analysis or metric used to group the data. Variations in this metric will result in different clusters, both in terms of the number and size. Therefore, knowing and selecting the method in which the clusters are formed is vital, as using one or the other will lead to variations in the results: Choosing a cluster formation method is equivalent to choosing a different clustering model.

3.1.1. K-Means

The K-Means [20] clustering model is one of the simplest and most popular models of the unsupervised learning paradigm. Its use in all types of applications and branches of knowledge has made this model one of the most widely used and best known.

The fundamental idea of K-Means lies in correctly locating what are called centroids, a reference where the data of a set can be compared. Various data that we want to group together will be compared with these centroids and, depending on how similar the data are, they will be grouped with one or the other within a cluster.

It should be noted that each cluster has only one centroid; therefore, a centroid can be seen as a centre of masses.

To measure the distance between each point and the centroids, two methods can be used [21]:

- The cosine distance between two points is calculated using the angle between the vectors obtained from them. Because X is a m x n data matrix that can be decomposed in m 1 x n row vectors $(x_1, ..., x_m)$, the cosine distances between vectors x_s and x_t are as follows.

$$d = 1 - \frac{x_s x_t'}{\sqrt{(x_s x_s')(x_t x_t')}} \tag{1}$$

- The Euclidean distance between points a and b is computed as follows.

$$d = \sqrt{\sum_{j=1}^{k} (a_j - b_j)^2} \tag{2}$$

The K-Means algorithm (Algorithm 1) is based on an iterative process, where actions will adjust the clusters until a state of convergence is reached, where the final results of the clusters formed are given. The steps are followed in an iterative manner to produce the clusters, and they are provided as follows [22].

As inferred from the way K-Means works, it is necessary to preset in advance how many clusters we want to form. This decision is crucial when running the model, as a misconfiguration of the number of clusters can lead to bad clustering either by default or by excess. In the limit, you cab see that a single cluster is generated with all data grouped in the same cluster or that each datum is the centroid of its own cluster. In both cases, the results would be useless. This can be extended to other hyperparameters, such as choosing a metric for distances between data or defining a good tolerance. One point to stress is the need to normalise the data before running it through the model, as K-Means uses distances to generate clusters.

Algorithm 1 K-Means

```
1: Prefix the number of clusters ($k$).
2: Randomly choose k centroids among the poins from the dataset ($D$).
3: $\forall x \in D \rightarrow$ Measure the Euclidean or Cosine distance to the centroids.
4: $\forall x \in D \rightarrow x \in$ nearest cluster.
5: Mean of each cluster = new centroids.
6: Run 3, 4 and 5 again until the variation of the new value of the centroid related to the
   previous iteration does not vary (or varies less than a preset tolerance).
```

The failure to normalise it results in data with high numerical values being weighted more heavily than others and vice versa, resulting in poor measurements and therefore poor clustering.

3.1.2. Hierarchical Clustering

Hierarchical clustering is another method for generating clusters [20]. It is based on the formation of clusters in an ordered, sequential and hierarchical way. This procedure can be approached from two points of view: by agglomeration or by division. In the former, it is assumed that all data form a single cluster and, at each iteration of the algorithm, similar clusters are merged until the number (k) indicated is reached. The behaviour of the algorithm is as follows (Algorithm 2):

Algorithm 2 Hierarchical Clustering by Agglomeration

```
1: $\forall x \in D \rightarrow$ x is an individual cluster.
2: Construct the distance matrix ($M$) with the Euclidean distances of each pair of clusters.
3: The most similar pair of clusters is merged.
4: Update M with the new cluster and the distances to the others.
5: Iterate points 3 to 4 until the pre-set number of clusters ($k$) is reached.
```

In hierarchical clustering by division, similarly to the previous procedure, it is assumed that all data form a single cluster. For each iteration, the cluster will be split into two clusters until the desired number of clusters is reached (k). The steps involved in hierarchical clustering by partitioning are as follows [23] (Algorithm 3):

Algorithm 3 Hierarchical Clustering by Division: Bisecting K-Means

```
1: $\forall x \in D \rightarrow$ x is an individual cluster.
2: The sum of Squared Errors (SSE) is calculated for each cluster.
3: The highest value is chosen and divided into clusters by K-Means.
4: Items 2 and 3 are iterated until the desired number of clusters ($k$) is reached.
```

3.1.3. Gaussian Mixture Modelling (GMM)

A Gaussian mixture modelling algorithm (GMM) [24] is an expectation-maximization (EM) type of model, and it is an iterative method where the aim is to find the maximum likelihood of a model by optimising the parameters that govern it. It is assumed that all points in the dataset are part of a mixture of normal distributions for their parameters are unknown. This algorithm, unlike previous ones, performs the classification in a soft way; that is, it does not assign a probability for belonging to each of the clusters of the set, and it is able to belong to several at the same time.

The clusters formed with GMM are assumed to follow a Gaussian shape, which is determined by two parameters (the mean (μ) and the standard deviation (σ or Σ)), in order to be maximised. To do so, the EM algorithm will be used, and it comprises two parts:

- Step E for calculating the probability that a point (x_i) belongs to each cluster, c_k:

$$r_{ic} = \frac{\text{Prob. of } x_i \text{ belonging to c}}{\text{Sum of prob. of } x_i \text{ belonging to } c_1, c_2, ..., c_k} \tag{3}$$

- Step M for updating the values μ, Σ, and Π (representing the point density of a distribution):

$$\Pi = \frac{\text{Number of points assigned to a cluster}}{\text{Total number of points}} \tag{4}$$

$$\mu = \frac{\Sigma r_{ic} x_i}{\text{Number of points allocated to a cluster}} \tag{5}$$

Both steps will be performed iteratively, optimising the parameters and maximising the associated likelihood function.

In a nutshell, the GMM algorithm (Algorithm 4) works as follows:

Algorithm 4 GMM

```
1: Number of clusters ($k$) selected.
2: Set randomly parameters of the different distributions.
3: Calculate the likelihood of the Gaussians with the data in the dataset.
4: Maximise the log-likelihood function by optimising parameters.
5: Iterate steps 3 and 4 until the indicated number of iterations is completed or a given
   tolerance is reached.
```

Although it is not as fundamental as previous algorithms, standardising and/or normalising the data before it is passed to the model can provide substantial improvements in its performance.

3.2. Size Reduction

Size reduction is the process required to analyse the importance of every feature of the data and to select those with effects on the calculations or those that are remarkable in the visualisation of the clusters without losing any significant information. In our context and to facilitate the visualisation and analysis of the data, it is necessary to reduce the number of features and, to this end, we have studied the following models. They will be evaluated in Section 6.

3.2.1. Principal Component Analysis (PCA)

Principal component analysis (PCA) is a statistical method [25] that can also be used as a first approach for how much information each of the characteristics of the data can offer to a machine learning model; thus, it choose the number of dimensions that for reducing the data without losing too much information. This interpretation can be performed by studying the accumulation of variances that are resent in data features. The higher the variance, the more information the data feature offers and the higher the value of the accumulated variance.

This method seeks to find out the variables that are the most important. To achieve this, PCA calculates the eigenvalues and eigenvectors of the covariance matrix of the data and selects those eigenvectors with the highest eigenvalue. These vectors will be used as the basis for the projection of data to these dimensions. After the projection, we obtain dimensionally reduced data as a result.

To sum up, the algorithm follows the next steps (Algorithm 5).

Algorithm 5 PCA

```
1: Calculate covariance matrix ($C$).
2: Calculate eigenvalues and eigenvectors of $C$.
3: Select the $m$ eigenvectors with the highest eigenvalue ($m$, the number of dimensions to
   reduce $C$ to).
4: Project the data onto the selected eigenvectors.
5: **Result**: Data reduced to $m$ dimensions.
```

3.2.2. ISOMAP

ISOMAP is another dimensional reduction algorithm [26], which is part of what is called manifold learning, a mathematical space where Euclidean space is recreated locally but not globally. This implies that the points of the dataset that are distributed in the space will be conditioned by a hyperplane of a certain shape, which may prevent determining the distance between two points from necessarily following a straight line. For a more accurate description of the distance/similarity between two points, it is necessary to traverse the dimensional space of the manifold and measure their distances using a geodesic of that space.

When dealing with high-dimensional data, the assumption that the data lies within Euclidean space is not always true. Isomap therefore adopts the premise that the data to be reduced belongs to a manifold, so it will perform the dimensional reduction in a non-linear manner. This method will try to preserve the geodesics found in the manifold when projecting it in a lower dimension. To achieve this, Isomap will first create a graph with the shape of the manifold from clustering algorithms such as K-Nearest Neighbours (KNN). Once the network is formed, it calculates the geodesics from the distance of the nodes in the graph. Then, it uses eigenvectors and eigenvalues to make a projection on the eigenvectors with the highest eigenvalue and, thus, reduces it dimensionally.

Analysing the details of this method, the procedure to undertake the dimensional reduction is described as follows (Algorithm 6).

Algorithm 6 ISOMAP

```
1: Determine the neighbourhood of each point.
2: Construct a manifold graph, connecting each point to the nearest neighbour.
3: Calculate the minimum distance between two nodes of the graph using the Dijkstra
   algorithm, obtaining a matrix with geodesic distances of the points in the manifold.
4: Projection of the data: the distance matrix is squared, double centred, and the eigen-
   value decomposition of a matrix is computed.
```

As the complexity order of the process is $O(N^2)$, it will require large computational resources if the number of points is substantial. Therefore, MDS is normally used, and it is an algorithm that translates distances between points into a configuration of points mapped onto a Cartesian space. As a result, a representation of the points is presented, taking into account the possibility that they are contained in the manifold.

3.2.3. T-Distributed Stochastic Neighbor Embedding (t-SNE)

T-Distributed Stochastic Neighbor Embedding (t-SNE) is a non-linear dimensional reduction algorithm [27] intended for the visualisation of high-dimensional datasets. The effect of applying t-SNE to the data is to generate an algorithm that attracts similar data to each other and repels data that is not similar, such as electrical charges. The result is a form of clustering, where high-dimensional data are transformed and grouped with data that have been estimated as similar to each other at a lower dimensionality.

The procedure followed by t-SNE is described as follows (Algorithm 7).

Algorithm 7 t-SNE

```
1: Measure similarity of data in high-dimensional space: assign each point a Gaussian distribution with a given standard deviation. Points that are close to each other will have a high density value in that distributions, while distant points will have a low density value in that distribution.
2: Construct a similarity matrix in high-dimensional space.
3: Data are randomly projected from high-dimensional space to low-dimensional space.
4: The calculation of the similarity of the data in low-dimensional space.
5: Construct a similarity matrix in low-dimensional space.
6: Try to make low-dimensional matrix values as similar as possible to the high-dimensional matrix by applying the Kullback–Leibler divergence metric and gradient descent. This causes the grouping of similar points and they are separated from the rest of the points that are not similar.
```

As a result, a representation of the data is presented by taking into account the possible distribution that could occur in the high-dimensional space.

t-SNE has an associated hyperparameter (perplexity) [28] that determines the value of the standard deviation of the distributions used to perform the similarity calculation.

3.2.4. Uniform Manifold Approximation and Projection (UMAP)

Uniform Manifold Approximation and Projection (UMAP) is another dimensional reducer [29], and its main idea is to generate a graph from the data belonging to a high-dimensional space and to assign it to another graph that is, however, of low dimensionality.

The improvement made by this type of dimensional reducer [30] is the way in which it performs the construction of the high-dimensional graph: A radius of the variable size is spread over each point in the set. When the radii of two different points intersect, those points are connected.

The size of the variable radii is preset according to the density of the neighbourhood in which the datum is located.

The neighbourhood density is determined using the KNN algorithm or similar algorithms, and a determination is made in terms of which neighbourhoods are highly dense and which are sparsely populated.

Once the connections are made and the neighbourhood densities are known, data connections are weighted according to the neighbourhood's density.

Once these steps have been carried out, another graph is produced with respect to the number of the dimensions that we want the results in. Data with heavily weighted connections will stay together, while lightly weighted connections will tend to stretch and separate. These rotations can occur in projections made in the graph while keeping the structure of the data intact.

In UMAP, there are two fundamental hyperparameters that condition the outcome: the number of neighbours taken into account to create the high-dimensional graph and the minimum distance between point in the low-dimensional graph. The former controls how UMAP balances between preserving the local structures of the dataset and the global structures: low values of the number of neighbours will enhance the representation of local structures, while high values will enhance the representation of global structure. On the other hand, the minimum distance controls how tightly UMAP clusters the points, where low values will make the clustering narrower and high values will make the clustering wider, preserving the broad structure of the dataset.

3.3. Metrics

One of the key aspects when designing and evaluating a learning model is to check its performance, operation or accuracy. In order to know these characteristics of the model, it is necessary to carry out different tests to check the real performance of the algorithm, applying a metric that allows measuring a value or property to compare different models and represents a feature of the algorithm's operation or result.

However, estimating how well a machine learning model is working becomes complicated when working within the unsupervised learning paradigm. Despite not having a ground truth for comparisons, there are several metrics that can be used to infer how a model is working. The metrics shown below are mainly intended for clustering algorithms, which is the type of model used in this proposal.

3.3.1. Within Set Sum of Squared Error (WSSSE)

In a clustering algorithm, a first valid approach to determine how well clustering was executed would imply a visualisation in a scatter plot, although the type of dimensionality the data possess will have to be taken into account. If working data have more than three features, it would have to be dimensionally reduced before it can be plotted, and this process entails a loss of information.

However, there is a metric that allows us to know the total error of the distances of the data relative to the centroid to which it belongs; that is, we can know, for each point of the dataset, how far it is from the centroid of the cluster.

Mathematically, WSSSE [31] is represented as follows, where k denotes the cluster, S_k denotes the data from cluster k and j denotes the position of the vector.

$$\sum_{k=1}^{K} \sum_{i \epsilon S_k} \sum_{j=1}^{p} (x_{ij} - \overline{x}_{kj})^2 \tag{6}$$

The equation includes the following:

1. The sum of the squared error of the centroid components with the components of a data of its cluster.
2. The sum of the error of all the data of a cluster with its centroid, applying (1).
3. The sum of the total error for all clusters, applying (2).

The result is that we obtain the total error of all clustering procedures performed on the points. With this metric, the parameters affecting clustering processes, such as hyperparameter k, the distance metric, tolerance, etc., can be optimised. The aim is to reduce this metric as much as possible without overfitting.

3.3.2. Silhouette

It is another measure of how good a clustering model is. In essence, it is a metric that allows knowing the cohesion within a cluster as well as the separation with other clusters, thus measuring how well a datum is classified in a cluster. To obtain this information [32], the following distances need to be calculated:

- Mean distance between a datum with respect to all other points belonging to the same cluster: This distance is called the mean intra-cluster distance.
- Mean distance between a datum with respect to the rest of the points belonging to the next nearest cluster: This distance is called the mean nearest-cluster distance.

The range of values that this metric can take is between $[-1, 1]$, where high values indicate that the data are wel- assigned (high cohesion to its cluster and high separation with other clusters) and vice versa.

The mathematical expression representing this metric is as follows, where o denotes an observation, a denotes the mean intra-cluster distance and b denotes the mean nearest-cluster distance [33].

$$s(o) = \frac{b(o) - a(o)}{max\{a(o), b(o)\}} \tag{7}$$

3.3.3. Selection Criterion

One way to use WSSSE to obtain the optimal value of a hyperparameter is to study the 'elbow point' of the graph [34]. The idea is to plot graphically the variation of the WSSSE metric as we vary the hyperparameter of choice. From the resulting graph, the point at which a variation in the slope occurs is shown. That point defines the hyperparameter.

With respect to Silhouette, we want to select the hyperparameter that maximises its value without overfitting, so the selection criterion is similar to the one proposed in WSSSE. This metric provides more information on the goodness of clustering of a model than WSSSE. Thus, if the two metrics point to hyperparameters with different values, the one indicated by Silhouette is prioritised.

4. Proposal

For this development, we start from the need to control heterogeneous environments with data sources that stream in real time. This information flow can represent normal traffic or attacks and, therefore, needs to be analysed and classified upon receipt.

When there are different devices simultaneously emitting amounts of data in a secure environment, it is essential to have a system that reacts immediately to incoming data in the event that these logs contain potentially anomalous characteristics.

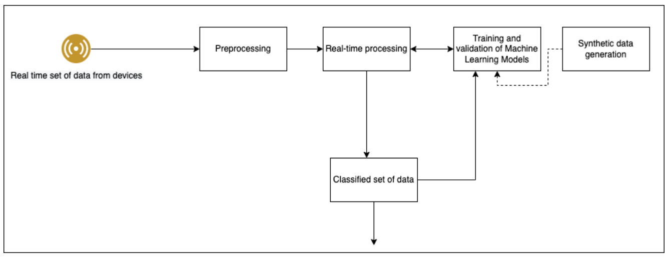
Due to the possible lack of logs in some of the devices that conform to a heterogeneous environment and the ability to extract features common to normal traffic, training the system to detect anomalous entries is difficult, so these features must be identified by a model that is previously trained to recognise normal traffic in the environment and, therefore, via unsupervised algorithms, these features automatically identify logs that do not resemble training logs. The definition of anomalies in this context is conditioned by these training data, because it is determined by a threshold of the Euclidean distance and takes into consideration the centre of each cluster and the position from which any data can be considered as anomalies.

This proposal follows the architecture described in Figure 1 and the structure below (Algorithm 8).

Algorithm 8 Proposal

```
1: Let $n$ be the number of devices analysed in an environment.
2: Let $i$ be a device from the set of $n$ devices analysed.
3: Let $F_i$ be the flow of information coming in real time from device $i$.
4: Let $m$ be the number of machine learning models trained.
5: Let $j$ be a model from the set of algorithms, applying function $M_j$.
6: $M_j(F_i) = \{0, 1\} \mid$ 0: normal traffic; 1: anomalous log.
```

In Figure 1, the pre-processing of the flow of information for selecting the best features that will be introduced to the model is shown, and the generation of synthetic data for devices that do not have enough data to be trained is also demonstrated. These processes will be explained in the context of the use case presented in the next section, where we introduce the solution proposed for the problem described.



Real time set of data from devices
Preprocessing
Real-time processing
Training and validation of Machine Learning Models
Synthetic data generation
Classified set of data

Figure 1. Systems involved in the architecture of the proposed model for anomaly classification (pre-processing, real-time processing, training and validation and synthetic data generation).

5. Designed Solution

As mentioned earlier, the aim of this study focuses on the use of unsupervised clustering algorithms to detect anomalies in real time. The main reason for using clustering algorithms is that, as seen in Section 2 in Related Work, they perform well in dealing with these types of situations. To achieve this, it is first necessary to provide a general introduction to the architecture (Figure 2), where one can observe how the entire workflow is constructed from the reception of the raw data to the classification of events by different models.

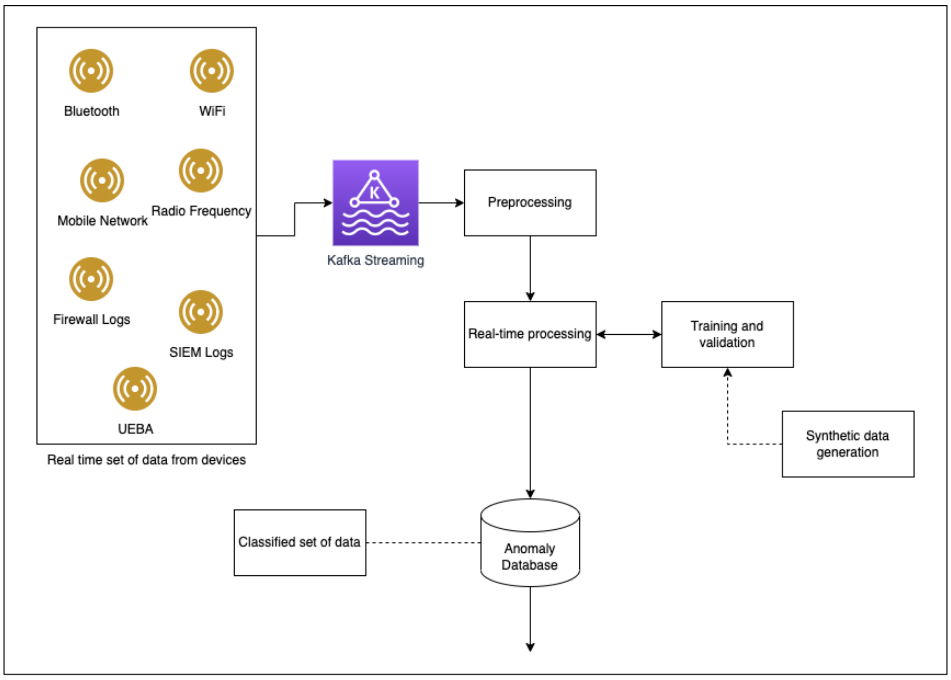


Figure 2. Modular architecture of the machine learning system.

Essentially, the machine learning system is responsible for detecting possible anomalies in the system's input data, which comes from heterogeneous sources. This information consists of sets of values in various formats that are fed into the Kafka Streaming module [35]. These data must be pre-processed by mathematical functions to transform the input values so that the machine learning algorithms embedded in the real-time processing subsystem and the training and validation subsystem can process them.

The machine learning algorithms must be previously trained with a set of data with the same characteristics as the data provided by the devices, and this is explained in Section 5.1.

In the training and validation subsystem, machine learning algorithms are trained. These algorithms are trained using various unsupervised methods. If existing datasets are not sufficient for training and validation, the synthetic data generation subsystem is used to generate a pair of normal/abnormal datasets so that the models can be properly trained and validated. Once machine learning algorithms have been trained, they are moved to the production phase. This is characterised by the real-time processing subsystem, which enables the processing of data flow from devices and identifies possible anomalies within the flow. The result is then stored in the anomaly database.

5.1. Training Dataset

The input interface corresponds to the amount of data collected by various devices from which data are received. The data sources are Mobile Networks, Radio Frequency, Bluetooth, WiFi, Firewall logs, SIEM logs and UEBA devices. The fields used from each of them are detailed below (Table 1).

Each of these modules sends the data it generates via the Kafka Flow Management Subsystem, and it uses differentiated topics (one per device), with each learning model subscribing to its corresponding topic to be handled.

Therefore, one unsupervised learning model is developed, trained and validated per topic (after selecting one of the possible models) and, thus, per the type of data source. Some of these devices have been previously tested in other projects [36,37].

Table 1. Data fields from each data source.

Device	Field	Description
Mobile Networks	Time	Device discovery time
	IMEI	Id of the detected device
	IMSI	International subscriber id
	RAT	Type of radio access (2G, 3G, and 4G)
Radio Frequency	Time	Date of measurement. EPOCH format, seconds resolution
	Signal	Signal strength level (dBms)
	Freq	Signal Frequency (MHz)
	mod	Signal modulation type (OOK, 2FSK, NONE)
	payload	Associated or extracted data from the signal
Bluetooth	Time	Creation time
	status	State the Bluetooth device is in
	classic_mode	If Bluetooth is in classic mode
	le_mode	Low Energy Bluetooth
	lmp_version	Bluetooth version used by the detected device
	address	MAC address of the detected device
WiFi	Time	Date of measurement. EPOCH format, seconds resolution
	userid	Identifier (MAC Address)
	footprint	Identifier in MAC format for random addresses
	tseen	Time (s) when the user has been detected in the measurement interval
	tacum	Time (s) that the user has been detected since the data source has been operational
	visits	Number of times the user has been detected after being disconnected (entries and re-entries)
	act24h	-
	pwr	The average power at which the device has been viewed during the measurement time
	tx_packets/tx_bytes	Number of packets/bytes transmitted in the measurement time
	rx_packets/rx_bytes	Number of packets/bytes received in the measurement time
	apwr	Average power of the access point seen by the device
	type	Type of MAC address (MAL, LMA, CID, and Unknown)

Table 1. *Cont.*

Device	Field	Description
Firewall Logs	Time	Date and time of measurement
	sequencenum	Number added to order logs with the same timestamp and origin
	source port	Source host port
	destination port	Connection destination port
	Xlate (NAT) source port	Source port after applying Hide NAT on the source IP address
	Xlate (NAT) destination port	Destination port after applying NAT
	VPN Peer Gateway	Main IP address of the VPN peer Security Gateway
	Blade	Product Name
	Action	Action of the matched rule in the access policy
	type	Log type
	interface direction	Connection direction
	source zone	Internal or External
	destination zone	Internal or External
	IP protocol	-
	needs browse time	Browse time required for the connection
	protocol	Protocol detected on the connection
	ICMP	ICMP message will be added to the connection log
	ICMP Type	For ICMP connection, type info will be added to the log
	ICMP Code	For ICMP connection, code info will be added to the log
	PPP	Authentication status
	Authentication method	Password authentication protocol used (PAP or EAP)
	scheme	Scheme used for the log
	methods	HTTP method
	VPN Feature	L2TP/IKE/Link Selection
SIEM logs	Date	Date of measurement. EPOCH format, seconds resolution
	Risk	Risk Level
	Signature	Description of the event created by SIEM
	Source	Source IP Address: <Port>
	Destination	Destination IP Address: <Port>
UEBA Data Source	**Field**	**Description**
Activity Tracker	clicks	Number of clicks made on the mouse
	pulsations	Number of clicks the user has made on the keyboard
	moves	Number of times the mouse has been moved around the screen
	scrolls	Number of times the scroll functionality has been used
Browsers	url	Web Address Visited
Files	path	Full address of the file on which the action was performed
	type	Type of action performed

Table 1. *Cont.*

UEBA Data Source	Field	Description
Network	bytes_sent/packets_sent	Number of bytes/packets that have been sent on that interface since the previous event
	bytes_recv/packets_recv	Number of bytes/packets received on that interface since the last event
	errin	Number of errors occurred while receiving packets since the last event
	errout	Number of errors occurred while sending packets on that interface
	dropout	Number of packets received that have been dropped since the last event
	name	Name of the supervised interface
Process	FinishTime	Date on which the process ended
	create_time	Date when the process started
	memory_usage	Maximum memory usage of the process during its lifetime (bytes)
	cores	Number of cores that the process is allowed to use
	cpu_usage	Maximum percentage of CPU used by the process during its lifetime
	nice	Priority of the process
	n_threats	Maximum number of threats created by the process during its execution
	childrens	Number of processes spawned by the process during its lifetime
Sockets	closed_time	Date and time of the measurement in EPOCH format (seconds resolution) at which the socket was closed
	detection_time	Date and time of the measurement in EPOCH format (seconds resolution) at which the socket was detected
	laddr/raddr	Local/Remote IP Address
	laddrport/raddrport	Local/Remote port number
	fd	Socket file descriptor
	type	Type of connection to which the socket is bound

5.2. Synthetic Data Generation

The synthetic data generation module is intended for the creation of synthetic datasets in models where the data do not meet the conditions of the model because certain devices cannot collect enough data inputs because they are not used in a real environment. In the use case of this study, the synthetic data generation module was used to obtain data similar to those produced by the Mobile Network data source.

The architecture of the module is shown in Figure 3.

The module uses the data provided from the inputs and establishes relationships between the attributes that are necessary for avoiding invalid inputs. In turn, profiles are defined to generate events by specifying the number of events that occur per clock step and then adding the probability that each profile will be executed per clock step.

This clock configuration is set in advance and specifies the simulation time and the time between steps. This configuration is conditioned by a time profile in which the characteristics of event generation are specified as a function of time (e.g., a higher number of steps during working time).

Finally, the attributes to be generated are defined with all configurations, relations and profiles. It should be noted that two different configurations are defined for the generation of a normal record and an anomalous record.

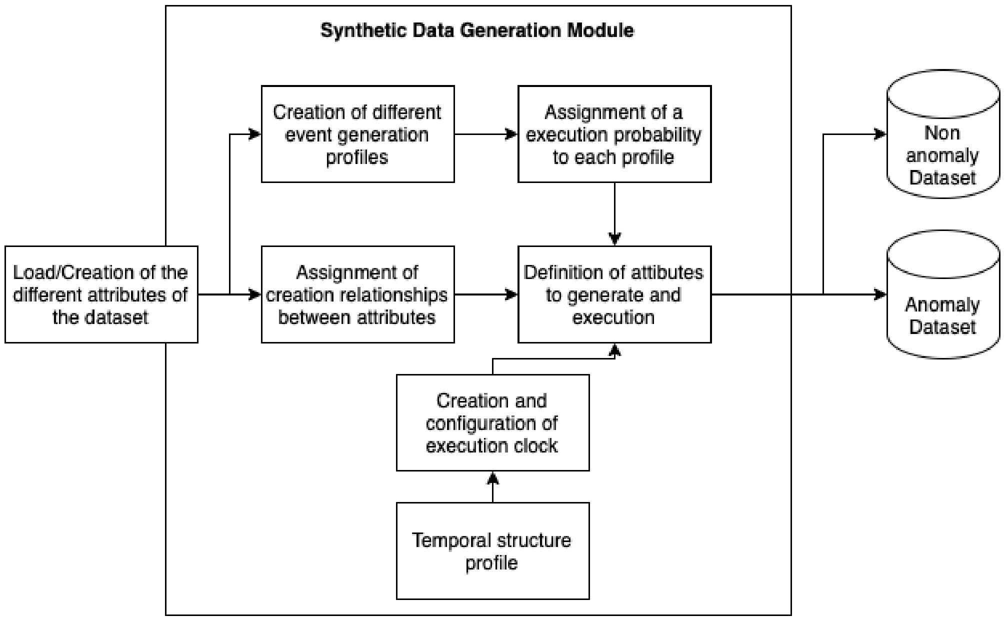

Synthetic Data Generation Module
Creation of different event generation profiles
Assignment of a execution probability to each profile
Load/Creation of the different attributes of the dataset
Assignment of creation relationships between attributes
Definition of attibutes to generate and execution
Creation and configuration of execution clock
Temporal structure profile
Non anomaly Dataset
Anomaly Dataset

Figure 3. Modular architecture of the synthetic data generation system.

All these configurations are related to the library for the generation of realistic datasets Trumania [38], which is a library that provides the necessary tools for the synthetic generation of data, the definition of the internal structures of the dataset and the prevention or enablement of the generation of events that meet the following requirements:

- Suitable types for all values of the dataset that facilitate their subsequent modification;
- Non-uniform structure for the generation of events;
- Plausible time structure in relation to real normal and abnormal scenarios;
- The impossibility of generating events that cannot occur in real environments.

The output data of the subsystem correspond to a pair of records generated with normal characteristics and a record with abnormal characteristics. The size of these created records is about 100,000 entries, and they can be varied in any order.

The characteristics of the generated datasets are identical to those generated and sent by the specific device with which we are up-sampling.

5.3. Preprocessing

This subsystem is responsible for normalising, transforming and standardising the values of the dataset before training the machine learning algorithms and real-time processing. Data preprocessing is defined in Figure 4 and divided into the different modules that make it up.

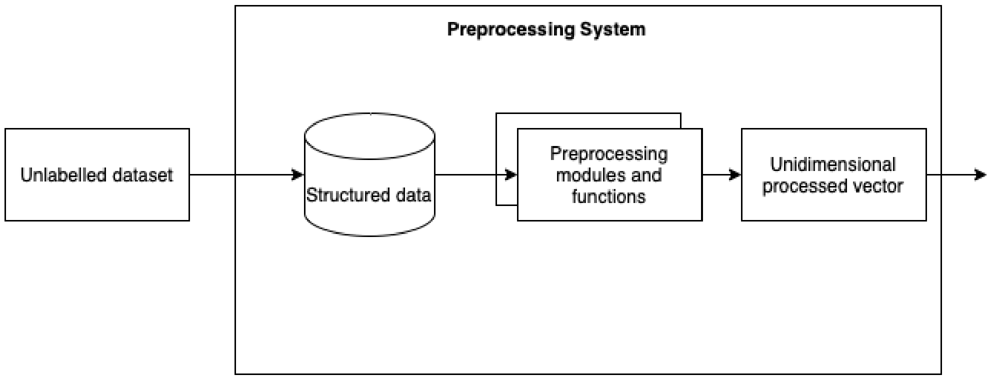

Preprocessing System
Unlabelled dataset
Structured data
Preprocessing modules and functions
Unidimensional processed vector

Figure 4. Modular architecture of the preprocessing system.

The input data of this system consist of a set of data values from each device. This information is first structured according to the type of datum so that the type of datum is defined. This structuring process is performed by defining a schema at the beginning of the data load that contains names of the attributes of the device data to which it refers, followed by the definition of the type of value contained in these attributes [39].

Then, preprocessing modules offered by Apache Spark and the respective functions defined for each type of event appear (String Indexer [40], Min-Max Scaler [41], Standard Scaler [42], One Hot Encoder [43], Hash Encoder [44], Regex Tokenizer [45], Count Vectorizer [46], TF-IDF [47], Word2Vec [48] and Vector Assembler [49]), performing relevant transformations and adjusting the data as a result. Some examples are presented in Figures 5–8.

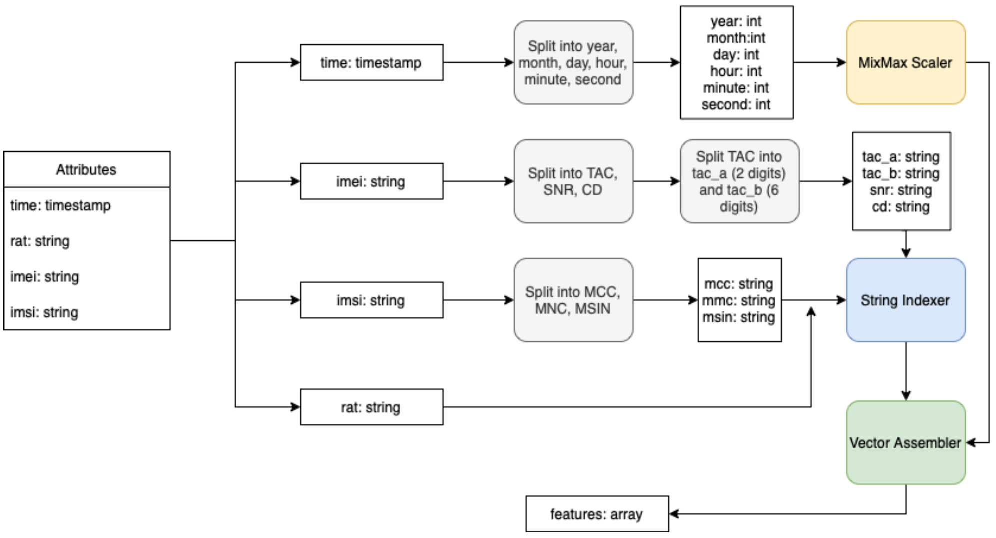

Attributes
time: timestamp
rat: string
imei: string
imsi: string
time: timestamp
imei: string
imsi: string
rat: string
Split into year, month, day, hour, minute, second
Split into TAC, SNR, CD
Split into MCC, MNC, MSIN
year: int month:int day: int hour: int minute: int second: int
Split TAC into tac_a (2 digits) and tac_b (6 digits)
mcc: string mmc: string msin: string
MixMax Scaler
tac_a: string tac_b: string snr: string cd: string
String Indexer
Vector Assembler
features: array

Figure 5. Preprocessing flow of Mobile Network device data.

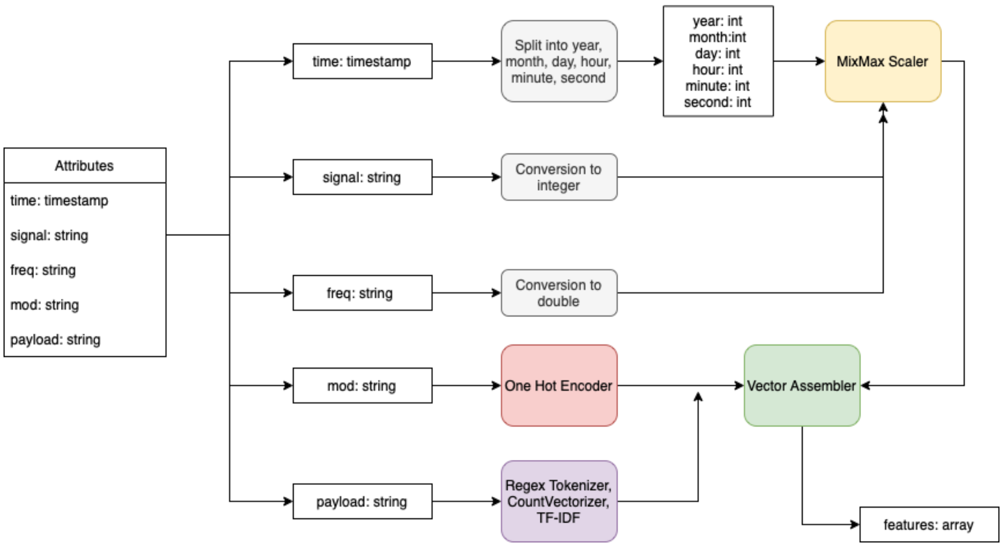


Figure 6. Preprocessing flow of Radio Frequency device data.

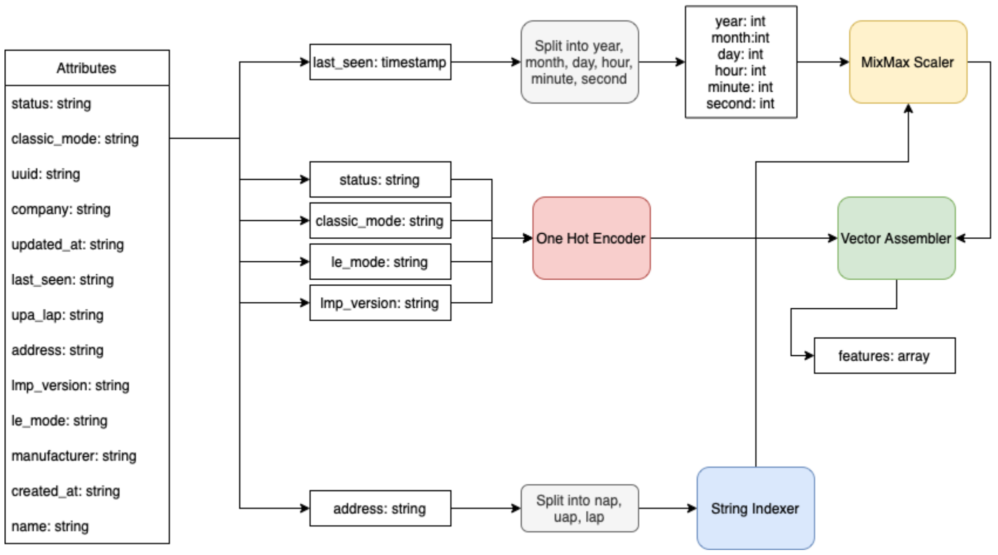


Figure 7. Preprocessing flow of Bluetooth device data.

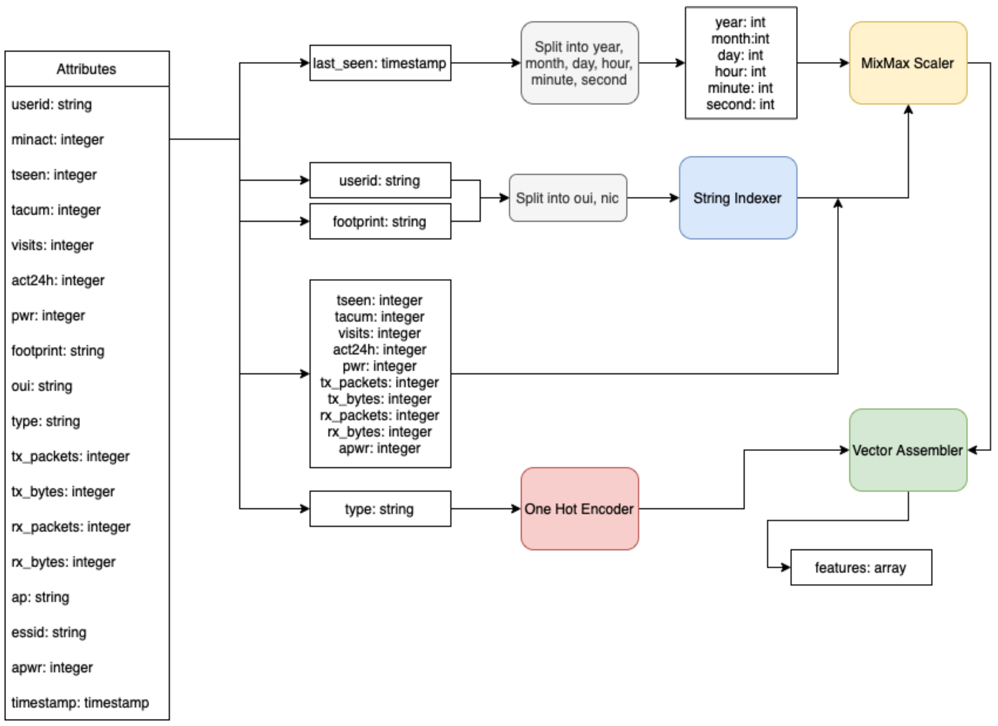


Figure 8. Preprocessing flow of WiFi device data.

5.4. Training and Validation

The training and validation subsystem aims to generate machine learning models that enable the detection of possible anomalies based on the data from the devices described above. The training and validation subsystem consists of the following components:

- Validation data;
- Training data;
- Hyper-parameter selection module;
- Machine learning algorithm selection module;
- Metrics module;
- Final model.

The internal architecture of this subsystem is shown in Figure 9.

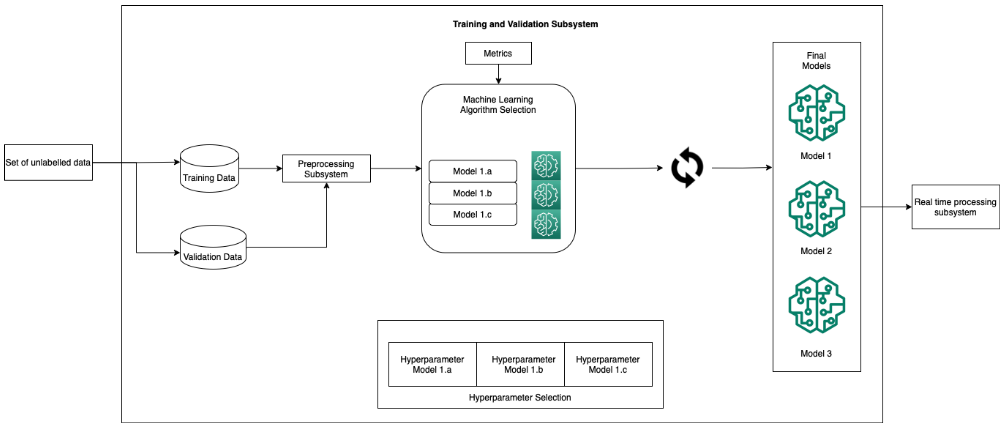


Figure 9. Modular architecture of the training and validation system.

The training and validation subsystem is responsible for preparing and monitoring the correct functioning of the machine learning model for anomaly detections. This is performed by using input data consisting of training data and validation data.

The training data are preprocessed by the preprocessing subsystem, which performs the above transformations depending on the type of event. For the identification of the best machine learning model for a given device type, hyperparameter selection modules, metrics and validation data are used.

The hyperparameter selection module is responsible for determining which feature set of a given machine learning model architecture is best suited for anomaly detection. Hyperparameters are selected based on the results provided by the metrics module. Therefore, the evaluation of the machine learning model for each set of devices is performed by the metrics module together with the validation dataset. This module allows the evaluation of the accuracy of an anomaly detection algorithm by applying a set of mathematical functions.

The mathematical functions chosen to determine the best hyperparameter for the model were the WSSSE metric and the Silhouette metrics used under the criterion of the 'elbow point' of the graph when selecting hyperparameters. It consist of plotting variations of the metric as we vary the hyperparameter value. From the graph, we select the point where the variation of the slope occurs.

Once the most accurate model has been obtained, the final model is determined for each device. The result, after going through the training and validation subsystem, is the different final models created for each device in the system.

5.4.1. Machine Learning Algorithm Threshold

The algorithms used to run the model to identify anomalies are the following: K-Means, bisecting K-Means and GMM. These algorithms allow the grouping of data into clusters of similar characteristics in an unsupervised manner.

The decision for one or the other algorithm is made after comparing the WSSSE and Silhouette metrics between the three models, as explained in the Results section.

The model with the best performance is selected, so the next step is to define the mechanism by which the model will detect an anomaly and a normal event. For this purpose, the models will use a threshold for each cluster formed. This allows us to define which data are sufficiently different from the rest of the cluster in order to be considered an anomaly.

Therefore, a model will be trained with a set of data that is considered non-anomalous so that, when new events are detected, whether these new data are similar enough to be detected as anomalies or not can be compared and observed .

The threshold is variable and can be more or less restrictive as needed. However, when defining the threshold, the default is to assign the distance greater than the furthest point from each centroid of the cluster to which it belongs.

In summary, the anomaly detection method works as follows:

- Assigning the optimal hyperparameters for each model depending on WSSSE, silhouette and training time;
- We compare the results of the metrics with each other and select the one with the best result;
- The selected model is trained with the data from the dataset of events considered normal, creating different clusters and setting a threshold for each;
- By default, the threshold is equal to the furthest point with respect to the centroid of the cluster to which it belongs. A threshold value is set for each cluster created. The value of the threshold can be changed as needed.

5.4.2. Selection of the Hyperparameters

The selection of hyperparameters in machine learning algorithms is crucial when designing the models to be implemented for each type of problem.

The method for choosing each hyperparameter was performed iteratively, testing a range of possible values and selecting the one that showed the best performance with the minimum number of clusters. This performance is represented in the WSSSE and Silhouette curves resulting from the iterative process performed on each hyperparameter and in the training time required for each of the models to be compared.

The metric that will be given more importance is Silhouette since it shows both the compactness of a cluster and the separation between clusters, while WSSSE only allows viewing the dispersion of the points of the clusters. In cases of similar Silhouettes, WSSSE will be observed and, in case it does not vary, training times will be analysed. In this paper, when analysing clustering algorithms, we will define the following hyperparameters in the three studied models. The results obtained are presented in Table 2 for models with optimisations that can be completed.

Table 2. Results of hyperparameters from each data source.

Device	Model	Hyperparameter	Value
Mobile Networks	K-Means	Nº of clusters (k)	2
		Distance Measurement	Euclidean
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
	Bisecting K-Means	Number of clusters (k)	-
		Distance Measurement	-
		Max Iterations	-
	GMM	Number of clusters (k)	3
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
Radio Frequency	K-Means	Nº of clusters (k)	2
		Distance Measurement	Cosine
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
	Bisecting K-Means	Number of clusters (k)	2
		Distance Measurement	Euclidean
		Max Iterations	By default: 100
	GMM	Number of clusters (k)	3
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}

Table 2. *Cont.*

Device	Model	Hyperparameter	Value
Bluetooth	K-Means	N° of clusters (k)	2
		Distance Measurement	Cosine
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
	Bisecting K-Means	Number of clusters (k)	2
		Distance Measurement	Cosine
		Max Iterations	By default: 100
	GMM	Number of clusters (k)	2
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
WiFi	K-Means	Number of clusters (k)	3
		Distance Measurement	Euclidean
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
	Bisecting K-Means	Number of clusters (k)	-
		Distance Measurement	-
		Max Iterations	-
	GMM	Number of clusters (k)	2
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
Firewall	K-Means	Number of clusters (k)	13
		Distance Measurement	Cosine
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
	Bisecting K-Means	Number of clusters (k)	9
		Distance Measurement	Cosine
		Max Iterations	By Default: 100
	GMM	Number of clusters (k)	4
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
SIEM	K-Means	Number of clusters (k)	17
		Distance Measurement	Cosine
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
	Bisecting K-Means	Number of clusters (k)	-
		Distance Measurement	-
		Max Iterations	-
	GMM	Number of clusters (k)	-
		Max Iterations	By default: -
		Tolerance	By default: -

Table 2. *Cont.*

Device	Model	Hyperparameter	Value
UEBA—Activity Track	K-Means	Number of clusters (k)	2
		Distance Measurement	Euclidean
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
	Bisecting K-Means	Number of clusters (k)	8
		Distance Measurement	Euclidean
		Max Iterations	By default: 100
	GMM	Number of clusters (k)	2
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
UEBA—Browsers	K-Means	Number of clusters (k)	7
		Distance Measurement	Euclidean
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
	Bisecting K-Means	Number of clusters (k)	7
		Distance Measurement	Euclidean
		Max Iterations	By default: 100
	GMM	Number of clusters (k)	9
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
UEBA—Files	K-Means	Number of clusters (k)	5
		Distance Measurement	Cosine
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
	Bisecting K-Means	Number of clusters (k)	3
		Distance Measurement	Cosine
		Max Iterations	By default: 100
	GMM	Number of clusters (k)	6
		Max Iterations	By default: 100
		Tolerance	By default: 700
UEBA—Network	K-Means	Number of clusters (k)	4
		Distance Measurement	Cosine
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
	Bisecting K-Means	Number of clusters (k)	4
		Distance Measurement	Cosine
		Max Iterations	By default: 100
	GMM	Number of clusters (k)	8
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}

Table 2. *Cont.*

Device	Model	Hyperparameter	Value
UEBA—Process	K-Means	Number of clusters (k)	5
		Distance Measurement	Cosine
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
	Bisecting K-Means	Number of clusters (k)	4
		Distance Measurement	Cosine
		Max Iterations	By default: 100
	GMM	Number of clusters (k)	7
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
UEBA—Sockets	K-Means	Number of clusters (k)	7
		Distance Measurement	Cosine
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}
	Bisecting K-Means	Number of clusters (k)	-
		Distance Measurement	-
		Max Iterations	-
	GMM	Number of clusters (k)	8
		Max Iterations	By default: 100
		Tolerance	By default: 10^{-4}

The high computational cost of training with the bisecting K-Means clustering model automatically implies that it is rejected as a solution for this proposal.

5.5. Real-Time Processing

The real-time processing subsystem is responsible for processing data from the various devices and identifying possible anomalies. These anomalies are determined by machine learning algorithms that were previously obtained by the training and validation subsystem. The internal architecture of this subsystem is shown in Figure 10.

The input data of the real time processing subsystem correspond to the set of data generated in real time by the set of devices. These input data are subscribed to Kafka, which manages the input stream data. This information is then sent to the pre-processing system, which is responsible for converting the input data into structured data in order to understand it properly.

Machine learning algorithms have been previously trained and validated in the corresponding subsystem to detect possible anomalies in the input data. In the real-time processing subsystem, there are several machine learning models for each type of device. The output data correspond to the result of the detection of normal or anomalous events. The trace with the values of the event itself and the result of the detection is stored in the "Anomalies" database with a unique identifier corresponding to the event.

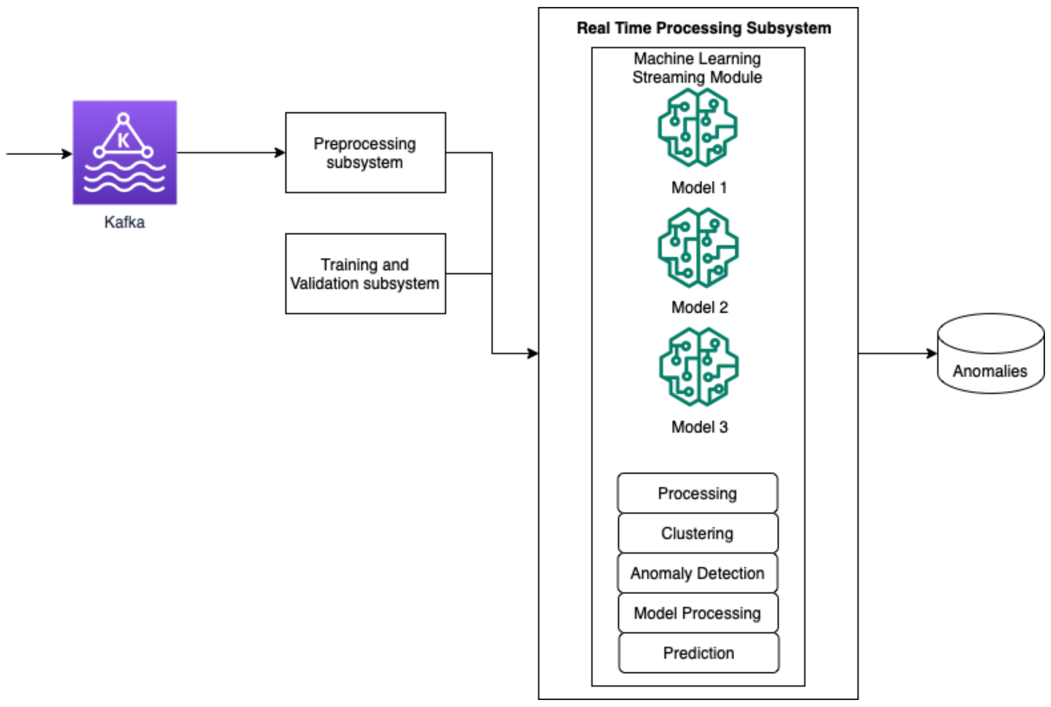


Figure 10. Modular architecture of the real time processing system.

6. Results

6.1. Model Comparison

The comparison of the three models associated with each of the devices is displayed after selecting the hyperparameters for each of them. The purpose of this comparison is to select the best model for each data source, taking into account the three metrics used: WSSSE, Silhouette and training time. For each of the three models per device, the same dataset and hardware/software environment was used to consider only the aspects related to the performance of the models themselves.

From the results presented in Table 3, it appears that the K-Means model performs equally or similarly to the bisecting K-Means model in most cases, but improves training time significantly by a factor of 10.

The GMM model appears to be the worst. Most tests, given the results of the Silhouette metric, show that its performance is significantly inferior to both K-Means and bisecting K-Means. This can be explained by the fact that GMM tends to get caught in local minima, resulting in those Silhouette values. Nevertheless, the result in training time is similar to K-Means. Another advantage of K-Means over the other two models is that it does not suffer from convergence or training time problems that prevent its use in some models, as is the case with GMM and bisecting K-Means. So, the model that seems to work best for each device is K-Means; therefore, this model is used for anomaly detection.

6.2. Detection Ratio

To see how the system works and how powerful it is, three types of tests were conducted for each model connected to each device.

Seven different data sources were considered that provided activity data to be tested as follows:

- Identification as normal data when the log represents non-anomalous activity;
- Identification as possible anomalous data if information has never been seen before and it is sufficiently different from the training data.

Table 3. Comparison of the different models for each device.

Device	Model	WSSSE	Silhouette	T. Time
Mobile Networks	K-Means	87,393.66	0.61	5.39s
	Bisecting K-Means	-	-	>>60s
	GMM	None	0.14	22.5s
Radio Frequency	K-Means	3.38	0.815	1.01s
	Bisecting K-Means	3.38	0.815	4.15s
	GMM	None	0.12	1.86s
Bluetooth	K-Means	3.47	0.62	0.91s
	Bisecting K-Means	3.47	0.62	7.01s
	GMM	None	0.57	0.82s
WiFi	K-Means	8571.55	0.50	1.13s
	Bisecting K-Means	-	-	>>60s
	GMM	None	0.2	2.81s
Firewall Logs	K-Means	1.99	0.71	3.35s
	Bisecting K-Means	3.89	0.61	25.9s
	GMM	None	0.14	3.53s
SIEM logs	K-Means	23.19	0.64	15.2s
	Bisecting K-Means	-	-	>>60s
	GMM	-	-	>>60s
UEBA Activity Track	K-Means	9.49	0.66	3.66s
	Bisecting K-Means	4.58	0.35	19.1s
	GMM	None	0.61	1.07s
UEBA Browsers	K-Means	0.87	0.76	0.89s
	Bisecting K-Means	1.01	0.66	18.7s
	GMM	None	0.62	1.53s
UEBA Files	K-Means	0.01	0.99	0.86s
	Bisecting K-Means	0.36	0.97	13.2s
	GMM	None	0.99	1.6s
UEBA Network	K-Means	5.03	0.96	0.75s
	Bisecting K-Means	5.07	0.96	11.6s
	GMM	None	0.94	1.63s

Table 3. *Cont.*

Device	Model	WSSSE	Silhouette	T. Time
UEBA Process	K-Means	6.72	0.88	0.78s
	Bisecting K-Means	8.01	0.86	0.78s
	GMM	None	0.69	1.64s
UEBA Sockets	K-Means	137.82	0.49	11.3s
	Bisecting K-Means	-	-	>>60s
	GMM	None	0.48	10.9s

For the test, two sets of samples are used for each device. The first set is considered as a normal set of samples and the second set of samples is considered anomalous.

Table 4 shows the result of measuring the accuracy of a system in an environment where normal data or data with possible anomalies have been defined. The result shows that almost all data considered normal and those with possible anomalies are classified as such.

It should be noted that all these results depend on the training and validation data provided/generated by each device. Although the data are promising, further testing is required to substantiate the results.

Table 4. Accuracy in detection.

Device	Normal	Anomaly
Mobile Networks	98.64%	96.00%
Radio Frequency	94.56%	98.00%
Bluetooth	99.32%	99.26%
WiFi	98.90%	99.30%
Firewall logs	97.98%	96.00%
SIEM logs	99.00%	98.75%
Activity Track	99.00%	99.30%
Browsers	98.00%	86.00%
Files	99.00%	97.63%
Network	96.00%	95.00%
Process	97.66%	99.60%
Sockets	98.90%	95.00%

6.3. Cluster Visualisation

Another way to determine if the cluster creation performance is correct is to visualise the different clusters created by different models. They need to be shaped correctly as they are used in the threshold for anomaly detection.

Once the clustering process is completed, in order to avoid any interference and only for visualisation purposes, it is necessary to perform a dimension reduction process to represent them in one plane. For this, four types of dimension reducers were used (PCA, ISOMAP, t-SNE and UMAP). In Figure 11, we show a visualisation where the clusters obtained with each model can be better identified after the dimension reduction process, abd we can verify that the anomalies are correctly detected by the model.

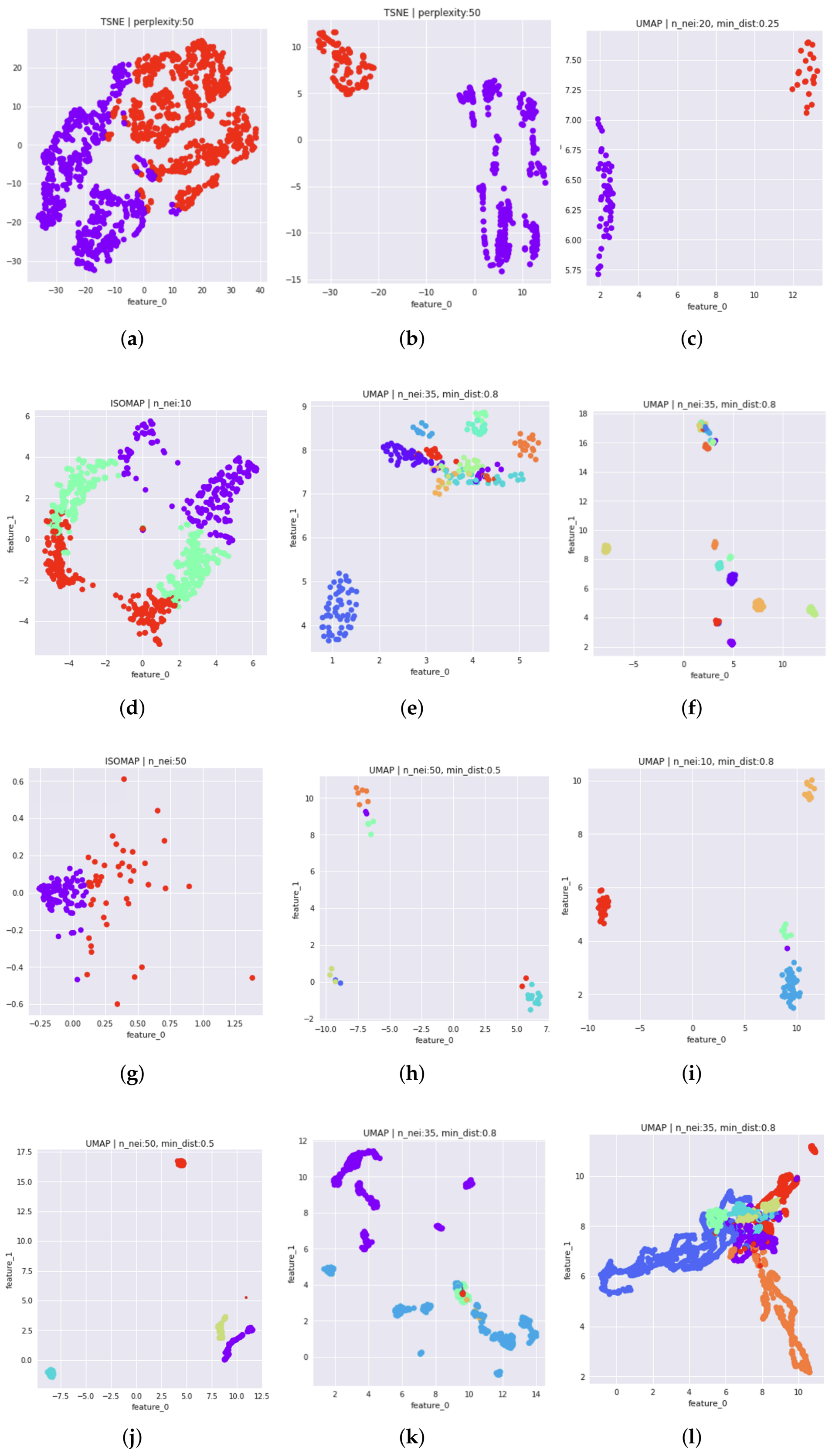


Figure 11. Clustering representation of the information from each device. (**a**) Mobile Network. (**b**) Radio Frequency. (**c**) Bluetooth. (**d**) WiFi. (**e**) Firewall Logs. (**f**) SIEM logs. (**g**) UEBA/Activity Track. (**h**) UEBA/Browsers. (**i**) UEBA/Files. (**j**) UEBA/Network. (**k**) UEBA/Process. (**l**) UEBA/Sockets.

6.4. Comparison with Related Works

In Section 2, we presented a relation of the previous works related to anomaly detection with different techniques in order to present the advantages of our proposal in relation to those results.

In Table 5, we included our characteristics to be compared with the Related Works, highlighting that our system obtained high quality results with widespread techniques and including real-time processing, a study and the pre-processing of data collected by our devices.

Table 5. Results obtained in previous studies

Research	Dataset	Preprocessing	Models	Real Time
This research	Own Datasets	Yes	K-Means	Yes
[7]	Own Dataset	Yes	K-Means	Yes
[8]	KDDCUP99 dataset	No	Mini Batch Kmeans with PCA	No
[9]	UCI Machine Learning Repository	No	K-Means and distance based approach	No
[10]	NSL-KDD dataset	Yes	SSC-OCSVM	No
[11]	Yahoo Webscope dataset	No	Statistical and deep-learning-based models	No
[12]	SWaT, WADI, SMD, SMAP and MSL datasets and internal dataset	Yes	Autoencoders	No
[13]	Aero-Propulsion System Simulation dataset	Yes	Isolated Forest	Yes
[14]	Own Datasets	No	AAE	No
[15]	Own Dataset	Yes	Isolation Forest Algorithm	Yes

7. Conclusions

Anomaly detection is a commonly addressed issue in recent cybersecurity research studies and is examined by various approaches. For the real-time processing of these incidents, unsupervised methods are very useful when the data that contain outliers are heterogeneous or specific for a certain device.

To do so, we have designed a real-time solution to deal with anomalies from various sources in a heterogeneous real environment.

After outlining all the modules and processes that make up the real-time anomaly detection system and conducting various tests with respect to its detection capability and performance, we were able to determine that the best algorithm among the proposed algorithms for tackling the threshold detection problem is K-Means, which is sometimes equivalent to bisecting K-Means, but it has better training times. GMM performed the worst and scored the lowest in the Silhouette metric. In the clusters obtained with K-Means, this configured threshold determines the difference between normal traffic and anomalies.

The conclusion drawn from the test conducted to check the system's ability to detect anomalies is that the detection of normal events and anomalous events had an acceptable performance, with an accuracy metric close to 99%. The UEBA–Browsers model had the lowest results, which was 86% for anomaly detection, while Process is the most accurate at identifying anomalous logs.

Finally, testing the performance of an unsupervised learning model is more complex than for other paradigms. The fact that events are undefined with respect to which are anomalous and which are normal is a major obstacle in determining the performance of the model. This means that the results shown may vary if the execution conditions or the configuration of different models change.

In the development of the real use case presented in this paper, we found some limitations, such as the impossibility to complete hyperparameter optimisation in some models because they exceeded the defined runtime for meeting the real-time condition or the lack of real logs for some of the models, which we had to synthetically generate. These issues may be addressed in the continuation of this line of research.

Moreover, for future developments, improving the treatment of possible abnormal events by temporal features is proposed. Since it is difficult to integrate this type of detection with the others, spliting the models (for each data source) into two models is proposed so that one can detect anomalies using non-temporal features and another can detect anomalies using temporal features. For better control over the process, we would also implement a blacklist/whitelist mechanism with the models to manually narrow down which hours are anomalous for the operator. On the other hand, we would like to test the result of combining the outputs of multiple models, especially the result of combining K-Means with GMM since one of the biggest problems of GMM is that it gets stuck in the local minima, which could be solved by first running K-Means and then initialising GMM with the centroids given by K-Means. Eventually, this proposal could be integrated into a cyber situational awareness platform.

Author Contributions: Conceptualization, C.S.-Z., X.L.-N. and V.A.V.; methodology, C.S.-Z., X.L.-N. and V.A.V.; software, C.S.-Z. and X.L.-N.; validation, C.S.-Z., X.L.-N. and V.A.V.; formal analysis, C.S.-Z., X.L.-N. and V.A.V.; investigation, C.S.-Z. and X.L.-N.; resources, C.S.-Z.; data curation, C.S.-Z. and X.L.-N.; writing—original draft preparation, C.S.-Z.; writing—review and editing, C.S.-Z., X.L.-N., V.A.V., M.S.R. and J.I.M.; visualization, C.S.-Z.; supervision, X.L.-N., V.A.V., M.S.R. and J.I.M.; project administration, V.A.V.; funding acquisition, V.A.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially supported by the Ministerio de Defensa of the Spanish Government within the frame of PLICA project (Ref. 1003219004900-Coincidente).

Institutional Review Board Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sarker, I.H. Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN Comput. Sci.* **2021**, *2*, 160. [CrossRef] [PubMed]
2. Apache Spark™—Unified Engine for Large-Scale Data Analytics. Available online: https://spark.apache.org/ (accessed on 9 December 2021).
3. Larriva-Novo, X.A.; Vega-Barbas, M.; Villagrá, V.A.; Sanz Rodrigo, M. Evaluation of Cybersecurity Data Set Characteristics for Their Applicability to Neural Networks Algorithms Detecting Cybersecurity Anomalies. *IEEE Access* **2020**, *8*, 9005–9014. [CrossRef]
4. ZANID HAYTAM. Outliers Detection in Pyspark #3—K-MEANS. Available online: https://blog.zhaytam.com/2019/08/06/outliers-detection-in-pyspark-3-k-means/ (accessed on 24 August 2022).
5. El ELK Stack: De los Creadores de Elasticsearch. | Elastic. Available online: https://www.elastic.co/es/what-is/elk-stack (accessed on 15 August 2022).
6. Jawale, A.; Magar, G. Survey of Clustering Methods for Large Scale Dataset. *Int. J. Comput. Sci. Eng.* **2019**, *7*, 1338–1344. [CrossRef]
7. Zhang, T.; Li, H.; Xu, L.; Gao, J.; Guan, J.; Cheng, X. Comprehensive IoT SIM Card Anomaly Detection Algorithm Based on Big Data. In Proceedings of the IEEE International Conferences on Ubiquitous Computing & Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS), Shenyang, China, 21–23 October 2019.
8. Peng, K.; Leung, V.C.; Huang, Q. Clustering Approach Based on Mini Batch Kmeans for Intrusion Detection System Over Big Data. *IEEE Access* **2018**, *6*, 11897–11906. [CrossRef]

9. Erdem, Y.; Ozcan, C. Fast Data Clustering and Outlier Detection using K-Means Clustering on Apache Spark. *Int. J. Adv. Comput. Eng. Netw.* **2017**, *5–7*, 86–90.
10. Pu, G.; Wang, L.; Shen, J.; Dong, F. A hybrid unsupervised clustering-based anomaly detection method. *Tsinghua Sci. Technol.* **2021**, *26*, 146–153. [CrossRef]
11. Munir, M.; Siddiqui, S.A.; Chattha, M.A.; Dengel, A.; Ahmed, S. FuseAD: Unsupervised Anomaly Detection in Streaming Sensors Data by Fusing Statistical and Deep Learning Models. *Sensors* **2019**, *19*, 2451. [CrossRef] [PubMed]
12. Audibert, J.; Michiardi, P.; Guyard, F.; Marti, S.; Zuluaga, M.A. USAD: UnSupervised Anomaly Detection on Multivariate Time Series. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '20), Virtual Event, 6–10 July 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 3395–3404. [CrossRef]
13. Khan, S.; Liew, C.F.; Yairi, T.; McWilliam, R. Unsupervised anomaly detection in unmanned aerial vehicles. *Appl. Soft Comput.* **2019**, *83*, 105650. [CrossRef]
14. Rajendran, S.; Meert, W.; Lenders, V.; Pollin, S. Unsupervised Wireless Spectrum Anomaly Detection with Interpretable Features. *IEEE Trans. Cogn. Commun. Netw.* **2019**, *5*, 637–647. [CrossRef]
15. Mercader, P.; Haddad, J. Automatic incident detection on freeways based on Bluetooth traffic monitoring. *Accid. Anal. Prev.* **2020**, *146*, 105703. [CrossRef] [PubMed]
16. Palacio-Niño, J.; Galiano, F. Evaluation Metrics for Unsupervised Learning Algorithms. *arXiv* **2019**, arXiv:1905.05667.
17. Tkachenko, R.; Izonin, I. Model and Principles for the Implementation of Neural-Like Structures Based on Geometric Data Transformations. In *International Conference on Computer Science, Engineering and Education Applications*; Springer: Cham, Switzerland, 2018. [CrossRef]
18. Tkachenko, R. An Integral Software Solution of the SGTM Neural-Like Structures Implementation for Solving Different Data Mining Tasks. In *International Scientific Conference "Intellectual Systems of Decision Making and Problem of Computational Intelligence"*; Springer: Cham, Switzerland, 2021. [CrossRef]
19. Unsupervised Learning and Data Clustering | by Sanatan Mishra | Towards Data Science. Available online: https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a (accessed on 26 August 2022).
20. Roman, V. Medium. 12 June 2019. Available online: https://medium.com/datos-y-ciencia/aprendizaje-no-supervisado-en-machine-learning-agrupaci%C3%B3n-bb8f25813edc (accessed on 3 August 2022).
21. Bora, M.; Jyoti, D.; Gupta, D.; Kumar, A. Effect of Different Distance Measures on the Performance of K-Means Algorithm: An Experimental Study in Matlab. *arXiv* **2014**. arXiv:1405.7471.
22. K Means Clustering | K Means Clustering Algorithm in Python. Available online: https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/ (accessed on 5 August 2022).
23. Understanding the Concept of Hierarchical Clustering Technique | by Chaitanya Reddy Patlolla | Towards Data Science. Available online: https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec (accessed on 25 August 2022).
24. Gaussian Mixture Models | Clustering Algorithm Python. Available online: https://www.analyticsvidhya.com/blog/2019/10/gaussian-mixture-models-clustering/ (accessed on 11 September 2022).
25. Lavrenko and Sutton. IAML: Dimensionality Reduction. 2011. Available online: http://www.inf.ed.ac.uk/teaching/courses/iaml/2011/slides/pca.pdf (accessed on 15 August 2022).
26. Tenenbaum, J.B.; Silva, V.D.; Langford, J.C. *A Global Geometric Framework for Nonlinear Dimensionality Reduction*; Science: New York, NY, USA, 2001.
27. Van der Maaten, L.; Hinton, G. Visualizing data using t-SNE. *J. Mach. Learn. Res.* **2008**, *9*, 2579–2605.
28. Cao, Y.; Wang, L. Automatic Selection of t-SNE Perplexity. *arXiv* **2017**, arXiv:1708.03229.
29. McInnes, L.; Healy, J. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv* **2018**. arXiv:1802.03426.
30. Coenen, A.; Pearce, A. Understanding UMAP. Available online: https://pair-code.github.io/understanding-umap/ (accessed on 1 August 2022).
31. Finding the K in K-Means Clustering | The Data Science Lab. Available online: https://datasciencelab.wordpress.com/2013/12/27/finding-the-k-in-k-means-clustering/ (accessed on 5 September 2022).
32. Wei, H. How to Measure Clustering Performances When There Are No Ground Truth? Available online: https://medium.com/@haataa/how-to-measure-clustering-performances-when-there-are-no-ground-truth-db027e9a871c (accessed on 14 August 2022).
33. Chaudhary, M. Silhouette Analysis in K-Means Clustering. Available online: https://medium.com/@cmukesh8688/silhouette-analysis-in-k-means-clustering-cefa9a7ad111 (accessed on 15 August 2022).
34. Drakos, G. Silhouette Analysis vs. Elbow Method vs. Davies-Bouldin Index: Selecting the Optimal Number of Clusters for KMeans Clustering. Available online: https://gdcoder.com/silhouette-analysis-vs-elbow-method-vs-davies-bouldin-index-selecting-the-optimal-number-of-clusters-for-kmeans-clustering/ (accessed on 12 August 2022).
35. Apache Kafka. Available online: https://kafka.apache.org/documentation/streams/ (accessed on 13 September 2022).
36. Alvarez-Campana, M.; López, G.; Vázquez, E.; Villagrá, V.A.; Berrocal, J. Smart CEI Moncloa: An IoT-based Platform for People Flow and Environmental Monitoring on a Smart University Campus. *Sensors* **2017**, *17*, 2856. [CrossRef] [PubMed]
37. Vega-Barbas, M.; Álvarez-Campana, M.; Rivera, D.; Sanz, M.; Berrocal, J. AFOROS: A Low-Cost Wi-Fi-Based Monitoring System for Estimating Occupancy of Public Spaces. *Sensors* **2021**, *21*, 3863. [CrossRef] [PubMed]

38. Sv3ndk, Milanvdm, FHachez, Thomas-jakemeyn, Petervandenabeele. Trumania. 2020. Available online: https://github.com/RealImpactAnalytics/trumania (accessed on 12 August 2022).
39. Larriva-Novo, X.; Vega-Barbas, M.; Villagrá, V.A.; Rivera, D.; Álvarez-Campana, M.; Berrocal, J. Efficient distributed preprocessing model for machine learning-based anomaly detection over large-scale cybersecurity datasets. *Appl. Sci.* **2020**, *10*, 3430. [CrossRef]
40. StringIndexer—PySpark 3.3.0 Documentation. Available online: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.StringIndexer.html (accessed on 13 September 2022).
41. MinMaxScaler—PySpark 3.3.0 Documentation. Available online: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.MinMaxScaler.html (accessed on 13 September 2022).
42. StandardScaler—PySpark 3.3.0 Documentation. Available online: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.StandardScaler.html (accessed on 13 September 2022).
43. OneHotEncoder—PySpark 3.3.0 Documentation. Available online: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.OneHotEncoder.html (accessed on 13 September 2022).
44. FeatureHasher—PySpark 3.1.3 Documentation. Available online: https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.ml.feature.FeatureHasher.html (accessed on 13 September 2022).
45. RegexTokenizer—PySpark 3.1.3 Documentation. Available online: https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.ml.feature.RegexTokenizer.html (accessed on 13 September 2022).
46. CountVectorizer—PySpark 3.1.3 Documentation. Available online: https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.ml.feature.CountVectorizer.html (accessed on 13 September 2022).
47. IDF—PySpark 3.1.3 Documentation. Available online: https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.ml.feature.IDF.html (accessed on 13 September 2022).
48. Word2Vec—PySpark 3.1.3 Documentation. Available online: https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.ml.feature.Word2Vec.html (accessed on 13 September 2022).
49. VectorAssembler—PySpark 3.1.3 Documentation. Available online: https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.ml.feature.VectorAssembler.html (accessed on 13 September 2022).