Annals of Telecommunications
https://doi.org/10.1007/s12243-021-00893-5

A threat monitoring system for intelligent data analytics of network traffic

Lucas C. B. Guimarães[1] · Gabriel Antonio F. Rebello[1] · Gustavo F. Camilo[1] · Lucas Airam C. de Souza[1] · Otto Carlos M. B. Duarte[1]

Received: 5 May 2021 / Accepted: 9 October 2021

Abstract
Security attacks have been increasingly common and cause great harm to people and organizations. Late detection of such attacks increases the possibility of irreparable damage, with high financial losses being a common occurrence. This article proposes TeMIA-NT (ThrEat Monitoring and Intelligent data Analytics of Network Traffic), a real-time flow analysis system that uses parallel flow processing. The main contributions of the TeMIA-NT are (i) the proposal of an architecture for real-time detection of network intrusions that supports high traffic rates, (ii) the use of the structured streaming library, and (iii) two modes of operation: offline and online. The offline operation mode allows evaluating the performance of multiple machine learning algorithms over a given dataset, including metrics such as accuracy and F1-score. The proposed system uses dataframes and the structured streaming engine in online mode, which allows detection of threats in real-time and a quick reaction to attacks. To prevent or minimize the damage caused by security attacks, TeMIA-NT achieves flow-processing rates that reach 50 GB/s.

Keywords Machine learning · Big data · Security · Threat detection · Stream processing

1 Introduction

Cybercrime is one of the major challenges introduced by the exponential growth of the Internet. According to Cybersecurity Ventures [1], damages related to cyber-attacks are projected to reach US$6 trillion by 2021. Besides, the growth and popularization of areas such as Big Data and the Internet of Things pose even more significant challenges to cybersecurity. The introduction of billions of low-power computing devices connected to the network increases the impact of possible attacks, as these devices can be easily hacked and compromised on a large scale [2–4]. The large volume of data to be analyzed in real time also increases the complexity of classifying network traffic and detecting threats [5]. Finally, the average time to detect an attack is a crucial factor in the impact of cyber threats. More than a quarter of cyber-attacks take a long time before being discovered, with this time often ranging from weeks to months [6]. The late detection of an attack exponentially increases the risk of financial losses and the risk of irreparable damage. The long time is due to the need for human intervention in these situations, significantly affecting the efficiency of dealing with threats.

In the scenario in which security is a fundamental aspect, the need for systems capable of guaranteeing safe and reliable network use is increasing. Solutions based on Security Information and Event Management (SIEM) tools partially mitigate the problem by providing real-time network monitoring. This type of solution, however, is still highly dependent on the intervention of experts and is based on threat signature databases, therefore being inefficient in the detection of new attacks. Using machine learning algorithms for threat detection, on the other hand, automates the detection process and meets the required agility to prevent and mitigate network attacks. It is of utmost importance to select algorithms that perform well in the classification process, without harming accuracy and other evaluation metrics. Previously, our research group

✉ Lucas C. B. Guimarães
chagas@gta.ufrj.br

Otto Carlos M. B. Duarte
otto@gta.ufrj.br

[1] Grupo de Teleinformática e Automação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil

Published online: 30 October 2021

(GTA/UFRJ) proposed CATRACA [7], a tool that uses machine learning to detect threats in real time.

This paper proposes TeMIA-NT: Threat Monitoring and Intelligent Data Analytics of Network Traffic, an intelligent threat monitoring and detection system based on machine learning and distributed processing in clusters. TeMIA-NT proposes and develops an entirely new distributed processing system with significant improvements in machine learning processing optimization. Our proposal focuses on the intelligence, scalability, and performance required to process large volumes of data while optimizing multiple machine learning algorithms to meet the diversity of new attacks. To increase performance, TeMIA-NT implements distributed processing entirely in Scala language and uses dataframe structures, instead of the standard Resilient Distributed Datasets (RDD) structure on the open-source Apache Spark platform. TeMIA-NT offers many options for machine learning algorithms and the possibility of optimizing hyperparameters, allowing testing, selecting, and adjusting the parameters of the best algorithm for each type of scenario. We implement the offline threat detection using the structured streaming library, which allows flow processing in micro-batches, with fault tolerance and reduced intervals. Online threat detection uses the continuous processing mode, which enables our proposal to perform similar to a native stream processing tool.

The rest of the article is organized as follows. Section 2 presents papers with themes related to the article. Section 3 introduces the Apache Spark platform, as well as its data structures and its machine learning library. Section 4 presents the machine learning algorithms used during the performance analysis, as well as a brief look at their hyperparameters. Section 5 offers a detailed look at the network traffic dataset used to test the proposed system, while Section 6 presents the system's architecture and features. Section 7 presents and analyzes the performance tests and their results, and Section 8 presents the author's final considerations and concludes the work.

2 Related works

New challenges in the intrusion detection area arise due to the high volume of traffic, a large number of IoT devices, distributed denial of service attacks, and zero-day attacks [8–10]. To meet these challenges, the use of machine learning techniques to classify flows in real time became popular [11–13]. The classification of large volumes of data at high speeds available employs three main distributed processing platforms: Apache Spark, Apache Storm, and Apache Flink. The fundamental difference between the platforms is that Spark performs batch processing while the Storm and Flink platforms perform native flow processing.

The Open Security Operations Center (OpenSOC) [14] is an analytical security framework for monitoring large amounts of data. OpenSOC originated a new project, Apache Metron [15], that is a tool that comprises the acquisition of different types of data, distributed processing, enrichment, storage, and visualization of results. Metron allows the correlation of security events from various sources, such as logs of applications and network packages. For this purpose, the framework uses distributed data sources, such as sensors on the network, logs of security element events, and enriched data called telemetry sources.

Based on the Apache Spark Platform [16], there are the Apache Spot, Stream4Flow [17], and Hogzilla. Apache Spot is a project still in the incubation stage that uses telemetry and machine learning techniques for analyzing packages to detect threats. The Stream4Flow prototype uses the Elastic stack to view network parameters; however, it lacks the intelligence to perform anomaly detection. The Hogzilla tool provides support for Snort, SFlows, GrayLog, Apache Spark, HBase, and libnDPI, offering network anomaly detection. Hogzilla also allows visualizing network traffic, using Snort to capture packets, and obtaining features through deep packet inspection. Stream4Flow captures packets using IPFIXcol and only considers header information. In our work, we use the flowtbag software, which captures various flow statistics. In addition, our offline processing mode allows updates to the machine learning model, further promoting the detection of new threats.

CATRACA [7] is an Intrusion Detection and Prevention System (IDPS) previously developed by our research group (GTA/UFRJ). CATRACA uses the Apache Spark processing platform and a machine learning model for model construction and flow classification, providing information about the real-time classification through a dashboard. However, the only algorithm available for building classification models is the decision tree; this model also works with default hyperparameter values, with no hyperparameter tuning being performed. Furthermore, CATRACA does not use Spark's DataFrame API nor its structured streaming processing engine, not taking advantage of the performance improvements offered by the platform's latest data structures. Finally, CATRACA has only one mode of operation, not allowing the construction and analysis of models without running them on a real-time scenario.

We select the Apache Spark platform to develop the TeMIA-NT because it is the most adopted among the examined Big Data processing platforms. Spark offers more possibilities for machine learning algorithms and is the one with the largest active community. Nevertheless, to the best of our knowledge, TeMIA-NT is the only available system to use the recent structured streaming technology in micro-batch and continuous modes in Apache Spark, allowing the

user to choose between "exactly once" fault tolerance or lower latency depending on the processing method. The system also allows the selection of several machine learning algorithms and operates in both offline and online modes.

3 The apache spark platform

We use Apache Spark [16], a distributed processing platform for Big Data, providing an interface for programming in clusters with parallelism and fault tolerance, to develop the system in this paper. We chose the Spark platform given its efficiency, its great acceptance in the market, and because it has a wide library of machine learning algorithms. The platform also supports multiple programming languages, including Python, Scala, R, and Java.

The main feature of Apache Spark is how it processes data: all operations that involve reading and writing intermediate results are done in memory. Spark is efficient for applications that perform multiple data transformation iterations in a distributed environment, avoiding time-consuming disk operations [18].

The Spark platform provides several libraries, such as Spark Streaming for real-time flow processing and GraphX for parallel graph computing. Also, Spark provides MLlib, a library that implements parallelizable and efficient machine learning algorithms in a distributed environment, making the platform an option for classifying network traffic. We use algorithms from the MLlib library to do performance analysis, which creates the machine learning models used for traffic classification.

3.1 Data structures

Because of the growing impact of Big Data, Zaharia et al. designed and developed Apache Spark to provide enterprise-level distributed processing for large datasets [16]. The data structures used by Spark play an essential role in fast and efficient data processing, being responsible for its organization, management, and storage; they also provide functions and operations to make more efficient data processing.

3.1.1 Resilient distributed datasets

The first Spark data structure developed for distributed processing was resilient and distributed datasets (RDD). This structure is an immutable and, therefore, resilient dataset, partitioned in the cluster nodes. It can be operated by a low-level API, offering multiple transformations and functions. A crucial feature of this data structure is to provide computing resources in memory, providing the agility observed in Spark operations. Another essential feature is the use of lazy evaluation, which computes expressions or functions only when their results are needed, optimizing the execution time by avoiding unnecessary calculations. RDD also offers fault tolerance: each RDD can reconstruct lost data automatically, based on data within other nodes in the cluster. Since RDDs are immutable, they can be created or retrieved at any time, making data sharing and replication a simple process.

3.1.2 Dataframe and dataset

DataFrames and Datasets are the other data structures implemented by Apache Spark. These structures differ from RDD in that they are structured as tables in a relational database: RDDs do not specify rows and columns, thus queries in RDDs with a large number of records require longer periods to complete. On the other hand, DataFrames and Datasets follow a schema, which lists the columns and the information they contain. As the data implemented through DataFrames and Datasets are structured, Spark implements performance optimizations in terms of processing time and memory consumption through the Tungsten [19] and Catalyst Optimizer [20] projects.

These data structures act similarly, differing only in terms of type handling: Datasets implement a strongly typed API, while DataFrames implement an untyped API. An untyped API allows parsing errors to go unnoticed during compilation time. Differently, a strongly-typed API detects these errors at compile-time, reducing the possibility of errors occurring during the execution of the program. Since Python and R do not have compile-time type security, these languages only implement DataFrames.

3.2 MLlib library

The purpose of the MLlib [21] library is to allow the use of machine learning techniques on the Apache Spark platform, implementing them in an efficient and scalable way through a high-level API. These techniques include standard classification, regression, clustering, and collaborative filtering machine learning algorithms, such as decision tree, linear regression, k-means, alternating least square, among others.

The library also offers featurization methods, allowing the Apache Spark platform to carry out the preprocessing of datasets before machine learning methods are applied. The application includes techniques that reduce dimensionality and rely on both the selection and extraction of features. These methods also allow the transformation of those features, such as normalization.

MLlib also provides multiple utilities to facilitate data processing, including statistical methods used to obtain results in terms of evaluation metrics, such as accuracy and

AUC, and linear algebra methods. There are also methods responsible for optimizing the execution pipelines, allowing algorithms and models to be saved and loaded from memory as necessary.

4 Machine learning and hyperparameters

Since they possess different logics and assume different characteristics of the input data, machine learning algorithms present different results depending on the target problem. Therefore, it is important to evaluate which algorithms offer the best results for the analysis and classification of network traffic. As such, the following algorithms made available through the MLlib library were implemented in the proposed system: naïve Bayes, logistic regression, support vector machine, multilayer perceptron, decision tree, random forest, and gradient-boosted tree. This analysis must be done by each network administrator since different algorithms may offer better performance depending on the chosen dataset. To enable this analysis, multiple performance metrics can be used, such as accuracy, F1-score, and area under the curve (AUC).

Hyperparameter tuning is the optimization of machine learning models, obtaining the best set of hyperparameters of an algorithm for a given dataset. Hyperparameters are the parameters that determine the learning process, and thus are selected before the training, in contrast with regular parameters that are learned during the training such as weights and bias. It is an exhaustive process, since it requires multiple executions of the learning algorithm, each time changing a specific parameter.

Some hyperparameters are unique to a given algorithm, such as naïve Bayes's smoothing. However, it is common for algorithms to share several hyperparameters; this can be seen on decision tree–based ones, all of which include a hyperparameter for setting the maximum tree depth, as well as on iterative ones, which all include hyperparameters for the maximum number of iterations and the minimum threshold necessary for convergence.

Naïve bayes The naïve Bayes methods are a set of probabilistic classifiers that work through the application of Bayes' theorem. This theorem, given by

$$P(c|x) = P(c)\frac{P(x|c)}{P(x)} \tag{1}$$

indicates the probability that an event c will occur knowing that a given event x has happened. The parameters used by the equation are the a priori probabilities of c and x, as well as their likelihood. Since the algorithm performs the classification through a simple mathematical calculation, resulting in linear execution time, it is easily scalable for large datasets and several features. However, the accuracy obtained by this classifier may be lower than that obtained by other algorithms, since it assumes that analyzed elements are statistically independent, which may not be valid depending on the chosen dataset.

The implementation of this algorithm on the Apache Spark platform provides two hyperparameters for tuning: the model type and the smoothing value. The model types available are multinomial, complement, Bernoulli, and Gaussian. Multinomial models are commonly applied to datasets containing categorical data. Complement is an adaptation of the multinomial method, used to better deal with unbalanced datasets. Bernoulli assumes that the data follows a Bernoulli distribution; as such, each feature must have binary or boolean values. Gaussian models assume that the probability distribution of the records follows a Gaussian distribution, allowing the models to handle continuous data. In turn, the smoothing hyperparameter is used to handle record values not observed during model training. Finding a new value results in a probability of zero and, as all probabilities are multiplied in the Bayesian equation, the final probability is also zero. Thus, the objective of the smoothing parameter is to ensure that the probability of each record is always greater than zero. Setting smoothing to 1 implements the Laplace smoothing, which is used by default by Apache Spark. Setting the smoothing parameter to values less than 1, but greater than 0 implements what is known as the Lidstone smoothing.

Logistic regression Logistic regression is a statistical algorithm that seeks to model the probability of a phenomenon, using the *sigmoid* function as a discriminant function. The function curve generated returns the likelihood of the data to be positive or negative. Based on this, the algorithm estimates the probability of new inputs to be or not in a certain class, making the binary classification.

The hyperparameters provided by Apache Spark are elastic net parameter, regularization parameter, the maximum number of iterations, convergence tolerance, threshold, fit intercept, and standardization.

The elastic net and regularization parameters influence the regularization applied during the calculation of the algorithm. The purpose of regularization is to reduce the overfitting of the model, adding a penalty to the loss function. Setting the elastic net parameter to 0 results in the use of the $L2$ norm as a penalty, while setting this value to 1 results in the $L1$ norm; intermediate values result in the proportional application of both norms. The $L1$ norm is calculated by adding the absolute values for each feature, while the $L2$ norm is obtained by adding these values squared. The penalty is then multiplied by the regularization parameter: small values for this parameter can still result in overfitting, while values that are too large may result in underfitting

the model. The maximum number of iterations and the convergence tolerance define stop conditions for the execution of the algorithm. The convergence tolerance defines that the execution of the algorithm must be interrupted if the improvement between two iterations is less than the defined tolerance; the maximum number of iterations interrupts the algorithm if the tolerance is not reached after a certain number of iterations. The threshold defines the value that is used to classify the records as belonging to a certain class, being a value between 0 and 1. The fit intercept is a Boolean hyperparameter, which defines whether a constant should be added to the decision function. Finally, standardization is another Boolean hyperparameter, which defines whether the training features are standardized by the algorithm itself. A standardization method must be applied to the dataset if regularization is used, as regularization is significantly influenced by the values of the features of the training set [22]. This hyperparameter defines that the standardization is done by the algorithm itself; however, it can be set as $False$ if this step is performed during the pre-processing of the data.

Support vector machine The support vector machine (SVM) algorithm maps the training data in space and performs a binary classification defining a hyperplane, the decision boundary. This hyperplane is set to partition the space, aiming to maximize the separation margin between the closest points of each of the classes. Altogether a larger margin results in a better generalization of the model.

A fundamental aspect of this method is the definition of its kernel function, responsible for the mapping done in the feature space. There are several kernel functions, the most used of which are: radial base function (RBF), polynomial, hyperbolic tangent, and sigmoid. However, the SVM algorithm used by Spark presents the linear kernel as the only available option. Similar to logistic regression, the stop conditions of the algorithm are determined by the maximum number of iterations and the convergence tolerance. Also present is the hyperparameter that determines whether the training features are standardized in the pre-processing, or during training, as well as whether the fit intercept is used. Finally, the regularization parameter is also available, acting on the impact of the penalty; however, unlike SVM, logistic regression supports only the $L1$ norm.

Multilayer perceptron Multilayer perceptron is a neural network model that works by employing multiple perceptrons, which act as the network's "neurons", who are delegated the tasks of performing small calculations and forwarding their results to other perceptrons. The perceptrons are organized in layers, with each perceptron in one layer being fully connected to the perceptrons in the next layer. The first layer receives the input features from the dataset, while the last layer represents the classification results.

Each perceptron uses an activation function to connect with others, based on the results of the previous layers and the adjusted weights for each output connection. These activation functions are non-linear, allowing the acquisition of non-linear models, but increasing the time required to obtain the model. One of the most used activation functions is the logistic (or sigmoid) function.

Another technique used by the multilayer perceptron is backpropagation; this algorithm works by calculating the gradient of the loss function concerning each weight by the chain rule, iterating one layer at a time from the last layer to avoid redundant calculations of intermediate terms. In this way, it is possible to update the weights of each layer to minimize losses.

The main hyperparameters that must be defined when using the multilayer perceptron are the number of hidden layers and the number of neurons in each layer. Most classification problems can be solved efficiently with one or two layers while using many layers tends to result in considerably longer processing times for ever-lower returns. For the number of neurons in each layer, a commonly adopted method is to use a single hidden layer, with the number of neurons being equal to the average between the number of characteristics and the number of labels in the data set. Another relevant hyperparameter is the learning rate, also known as the step size. This value acts in updating the weights during the execution of the algorithm; using a very low learning rate can result in long run times and overfitting, while higher learning rates can result in models with lesser performance. Other hyperparameters provided by Apache Spark for this algorithm include the maximum number of iterations, as well as the minimum tolerance for optimization, similar to SVM and logistic regression.

Decision tree The decision tree algorithm builds a tree in which each internal node evaluates a data feature. Each branch represents a decision around a possible value for the selected feature, and each final node in a branch indicates the class the element is most likely to belong to. Thus, the algorithm traverses the tree branches and evaluates the features of each node to estimate the sample probability to belong to a particular class. A great advantage of the decision tree algorithm is its ease of understanding and interpretation, being composed exclusively by rules in the "if-then-else" format.

The most important hyperparameters provided by Apache Spark for the decision tree are the maximum depth of the tree, the minimum gain of information, the minimum number of instances per node, and the metric selected to calculate the impurity of each feature. The maximum depth controls the generalizability of the algorithm and

directly influences the training and test time of the algorithm. Extremely deep trees tend to divide the entire training set into their correct labels, and as a consequence, they are overfitted, while trees with few levels are unable to capture the variance present in the dataset and tend to have low classification performance. The minimum information gain hyperparameter is the value that must be obtained to consider the division of a given node, controlling the growth of the tree by restricting which nodes can be created to divide the dataset. The minimum number of instances per node controls the growth of the tree and determines the minimum number of samples required in the children of the node to generate the branch. The node of the tree that is not able to generate the minimum number of samples for the right and the left child becomes a leaf of the tree. A larger minimum number of samples can positively influence the model's accuracy for large datasets since a low number can lead the model to behave randomly. Impurity measures the diversity of children raised using characteristics that meet the criteria for division. Thus, impurity is a criterion for selecting among all candidates a feature with greater diversity to perform the division of the node. Impurity is measured using the Gini index or entropy, the main difference of which being the slower computation of entropy.

Random forest The random forest is an ensemble learning algorithm, proposed by Breiman [23], that works by creating multiple decision trees. Breiman also proposed the bagging method to create different decision tree structures and capture distinct behaviors of a dataset.

The bagging method comprises two phases: bootstrap and aggregating [24]. The bootstrap phase consists of generating equally-sized datasets from the original training dataset through random sampling. Then, the method trains decision tree models from each sampled dataset. The goal is to build learning models with different structures that present different views when classifying new samples. In the aggregating phase, the method uses all different model structures of each local model to discover the correct class of a new sample. Each machine learning model classifies the sample, and the final result is the statistical mode of all classifications. This way, the method can generalize the behavior of new samples while minimizing variance.

In a random forest [23] with H trees, the predicted class $\hat{y}$ of a sample x is given by:

$$\hat{y} = f(x) = arg\max_{y \in Y} \sum_{j=1}^{H} I(y = h_j(x)), \tag{2}$$

where $h_j(x)$ returns the predicted class of x by tree h_j. The term $I(.)$ is the indicator function. The set Y represents the existing classes, which in our work are binary: 0 for normal flows and 1 for malicious flows.

The number of trees in the forest and the subsampling rate are the adjustable hyperparameters for random forests, in addition to the hyperparameters of decision tree models. As the number of trees grows in the forest, the classifier's performance increases due to the high variance of the built decision trees. However, after approximately 100 trees the metrics remain statistically equal, only increasing the processing time [25]. The subsampling rate hyperparameter specifies the size of the dataset used to train each tree in the forest and is defined as a fraction of the size of the original dataset.

This algorithm usually presents better results than those obtained by working with only one decision tree, in addition to offering less risk of overfitting, but it has a considerably longer processing time. However, random forests are extremely parallelizable, since the training and classification of a single tree are independent of the set. The adoption of parallel processing reduces the complexity of the algorithm [26].

Gradient-boosted tree As well as the random forest algorithm, the gradient-boosted tree is an ensemble learning algorithm based on decision tree models. Unlike random forest, where each tree is trained individually, the gradient-boosted tree trains all trees iteratively, where the new trees use the prediction of previous trees to offer a more accurate model.

The gradient-boosted tree uses the Boosting method to optimize the model after each iteration. Several shallow trees are created, calculating the loss based on that tree created. In the next iteration, the algorithm creates a tree that aims to reduce the loss value generated by the previous function. This process is interrupted if the algorithm reaches a stop condition, such as the maximum number of trees created or if the next tree does not improve the model's metrics.

This algorithm also has a high processing time and is not ideal for large datasets. As it is a tree-based algorithm, it presents the same set of parameters and hyperparameters as the decision tree and random forest, except for the learning rate, also known as step size hyperparameter. The learning rate is the hyperparameter that controls how complex the next tree built will be, giving more relevance to the mistakes done by the previous tree.

4.1 Structured streaming

The real-time processing on the Apache Spark platform was initially implemented through the Spark Streaming library, which allows continuous processing of RDDs through the DStream API. With the introduction of DataFrame and Dataset as new data structures, the structured streaming library was developed to handle these structures in real time while maintaining the optimizations they introduced.

Structured streaming allows the programmer to program in a similar way to the one in batch data processing, with the platform dealing with the implementation of specific flow processing techniques through a high-level API. Structured streaming implements the micro-batch technique, with data received within a certain time interval being added to a batch to be processed; after processing, the result is added to a table, and the elements of the processed batch are discarded. Other advantages of the library include "exactly once" fault tolerance, as well as end-to-end latency of up to 100 ms.

Another processing method provided by the library is the continuous processing mode. This mode allows latency as low as 1 ms but does not offer all the functions of the main library, supporting only projection and selection operations. It also has "at least once" fault tolerance, leaving aside the advantages of tolerating exactly once of the other processing method.

5 Dataset and schema

A crucial aspect of the development of an intrusion detection system (IDS) is the need to check its performance before it goes into operation. Thus, a dataset is used that contains both legitimate and malicious traffic. The most commonly used dataset in IDS development is the NSL-KDD [27], with other important datasets being the DARPA98 and the DARPA99. However, these and other datasets are often not recent, and in addition to using synthetic attack patterns and threats, may not portray the features of current network traffic.

The dataset used was obtained from traffic from a telecommunications operator [28], converted into flows using the flowtbag tool. Each flow is a sequence of packets, within a time window, which has certain features in common. The features used to group packets in flows were the 5-tuple (source IP address, destination IP address, source port number, destination port number, protocol), set commonly used in traffic analysis works. After grouping packages into flows, the flowtbag tool extracts 40 features for the construction of the data schema, including the number of packages sent and received, the minimum and maximum sizes of a package, among others. The complete list of features, including the 5-tuple, is presented in Table 1.

In the preprocessing stage, we identified that the dataset had seven features that contained only null values; after these columns were removed, we calculated the Pearson correlation matrix shown in Fig. 1 to allow a better understanding of the remaining features. Through this matrix, it is possible to identify that multiple features possess a high correlation between them. For instance, features 7, 8, and 9 have a high correlation with feature 6, since all these features are related to the flow size; features 26, 27, 28, and 29 also have a high correlation, with these four features representing time measurements of the flow.

The dataset used groups together a series of attacks common on computer networks, such as attacks focused on the application, transport, and network layers. Most of the observed attacks occurred at the application layer, because even though each layer presents its vulnerabilities, the application layer allows less sophisticated attacks to occur, which can be performed by inexperienced attackers.

The labeling of the dataset flows as legitimate or malicious, necessary for the creation of models in supervised machine learning algorithms, was done through IDS Suricata. The dataset was also balanced to avoid bias during the model training and test phases, therefore being composed of equal parts of legitimate and malicious traffic.

6 The proposed architecture

The proposed system has two operation modes: online and offline. The online mode performs classification in real time, whilst the offline mode allows to observe the performance of multiple classifiers for a given dataset, making the resulting metrics available in the visualization module.

The proposed architecture, shown in Fig. 2, is modular and consists of three main modules: data collection, processing, and visualization.

The data collection module captures and abstracts network traffic flows. It also stores the datasets used in offline processing. The capture process reflects network traffic through the libpcap library. Then, the flowtbag tool abstracts the sequence of packets in the flows and their 40 features, including the flows length and the total number of packages for each flow. We use the five fields of the TCP/IP packet header, source IP address, destination IP address, source port, destination port, and protocol to abstract packets into flows. A channel on the Apache Kafka platform, which acts as a data buffer, receives the streams of data. We use the Hadoop Distributed File System (HDFS), a distributed database, to store the datasets used to train and test the classification models.

The processing module carries out the process of classifying these flows. The processing module is implemented in an Apache Spark cluster. This platform presents advantages to the development of the system, as it has libraries aimed at the implementation of machine learning algorithms and the fast processing of data in real time, using the micro-batch method with the structured streaming engine. The training module extracts the classification model using a dataset labeled from HDFS. In the online mode of operation, packages are collected and added to an Apache Kafka channel, and flows are then classified as legitimate or malicious

Table 1 Meaning of all network dataset rows generated by flowtbag [29]

No.	Name	Description
1	srcip	Source ip address
2	srcport	Source port number
3	dstip	Destination ip address
4	dstport	Destination port number
5	proto	IP protocol used for the connection
6	total_fpackets	Total packets in the forward direction
7	total_fvolume	Total bytes in the forward direction
8	total_bpackets	Total packets in the backward direction
9	total_bvolume	Total bytes in the backward direction
10	min_fpktl	Size of the smallest forward packet
11	mean_fpktl	Mean size of forward packets
12	max_fpktl	Size of the largest forward packet
13	std_fpktl	Standard deviation (SD) from the mean of the forward packets
14	min_bpktl	Size of the smallest backward packet
15	mean_bpktl	Mean size of backward packets
16	max_bpktl	Size of the largest backward packet
17	std_bpktl	SD from the mean of the backward packets
18	min_fiat	Minimum amount of time between two forward packets
19	mean_fiat	Mean amount of time between two forward packets
20	max_fiat	Maximum amount of time between two forward packets
21	std_fiat	SD from the mean time between two forward packets
22	min_biat	Minimum amount of time between two backward packets
23	mean_biat	Mean amount of time between two backward packets
24	max_biat	Maximum amount of time between two backward packets
25	std_biat	SD from the mean time between two backward packets
26	duration	Duration of the flow
27	min_active	Minimum time that the flow was active before idle
28	mean_active	Mean time that the flow was active before idle
29	max_active	Maximum time that the flow was active before idle
30	std_active	SD from the mean time that the flow was active before idle
31	min_idle	Minimum time a flow was idle before becoming active
32	mean_idle	Mean time a flow was idle before becoming active
33	max_idle	Maximum time a flow was idle before becoming active
34	std_idle	SD from the mean time a flow was idle before turn active
35	sflow_fpackets	Average number of packets in a forward sub flow
36	sflow_fbytes	Average number of bytes in a forward sub flow
37	sflow_bpackets	Average number of packets in a backward sub flow
38	sflow_bbytes	Average number of packets in a backward sub flow
39	fpsh_cnt	Number of PSH flags in forward packets
40	bpsh_cnt	Number of PSH flags in backward packets
41	furg_cnt	Number of URG flags in forward packets
42	burg_cnt	Number URG flags in backward packets
43	total_fhlen	Total bytes used for headers in the forward direction
44	total_bhlen	Total bytes used for headers in the backward direction
45	dscp	First set DSCP field for the flow

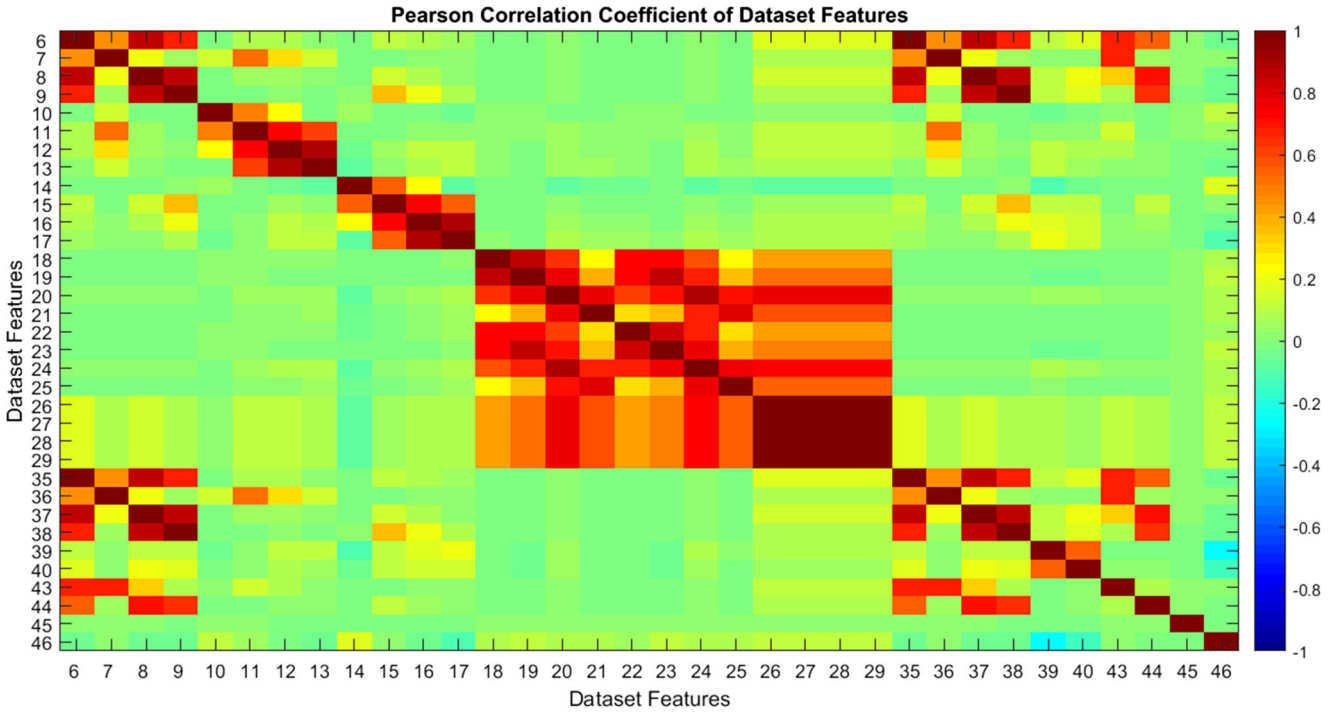


Fig. 1 Matrix of Pearson correlation coefficients of all features present in the dataset

by the classification model obtained previously. For execution in the offline mode, the classification module runs tests on various algorithms and datasets, obtaining performance metrics for each combination. After obtaining the result from either classification module, the data enrichment module collects additional data about the labeled flows, such as IP-based geolocation. The results of both online and offline classifications, as well as the additional data added by the data enrichment module, are then sent to an Elasticsearch server using the Apache Spark integration library.

The visualization module allows the network administrator to visualize the classifications history and the current state of the network, as well as the results of tested algorithms. We implement the visualization module using the Elasticsearch[1] and Kibana[2] software, both developed by Elastic. Elasticsearch implements a distributed and efficient search server, based on JSON documents. It receives and stores the data as the processing module sends it after the classification process is finished. Kibana is responsible for providing a user interface through dashboards, displaying to the network administrator the data received by Elasticsearch in real time for both execution modes. It also allows consultation by historical data, using the search server features Elasticsearch.

The proposed system only handles the detection of malicious traffic, not dealing with the mitigation of current attacks nor the prevention of future ones. Thus, the system must work together with mitigators to react and minimize the damage resulting from attacks. After identifying an attack, the flow and all extracted features can be sent to a mitigation service chosen by the network administrator; the mitigator can then use this information to better react to the attack.

While the system only identifies if the incoming traffic is potentially malicious, and does not specify the type of attack, the information provided by the proposed system can help identify what kind of attack is being executed, which will help in the mitigation and prevention process. For instance, if a Distributed Denial of Service (DDoS) attack is identified, and the network administrator has a prior agreement with a mitigation service that implements Distributed Denial-of-Service Open Threat Signaling (DOTS) [30], the additional information about malicious flows can be sent to the mitigation service through DOTS, aiding the mitigation of future attacks.

7 Performance analysis

A cluster of four computers, one master and three slaves, using Ubuntu 19.04 operating system, composes the performance analysis environment. The master is a

[1] https://github.com/elastic/elasticsearch, accessed in April 2021.

[2] https://github.com/elastic/kibana, accessed in April 2021.

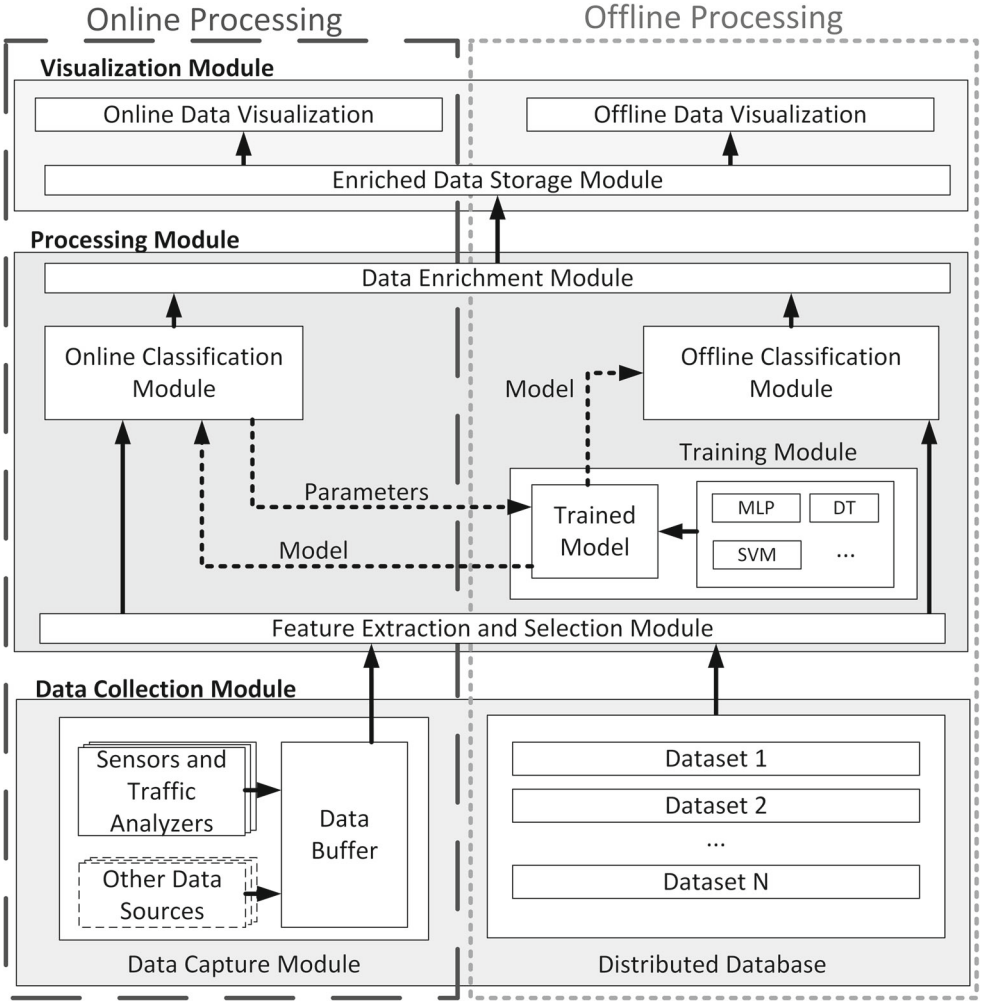


Fig. 2 TeMIA-NT modular architecture at online and offline modes

biprocessed Xeon X5570 with 4 cores and 96 GB of DDR3 RAM, and the slaves are biprocessed Xeon E5-2650 with 8 cores and 32 GB of DDR3 RAM.

The experiments use accuracy, precision, sensitivity, F1-score, and false negative rate (FNR) to evaluate the performance of the algorithms. Accuracy refers to the closeness of a measured value to a known value and precision refers to the closeness of two or more measurements to each other. Therefore, accuracy is given by the number of flows correctly classified divided by the total number of flows. High accuracy means that positively rated flows are less likely to be negative. Precision calculates the ratio of positive flows correctly classified among all flows classified as positive. Sensitivity calculates the proportion of all positive flows correctly classified among the actual positive flows. A high sensitivity means that most of the real positive flows have been classified correctly. The F1-score is the harmonic mean of precision and sensitivity, being a metric that takes into account false negatives and positives. The false negative rate detects the number of false negatives, being equal to 1 minus the recall. Their equations are

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

$$F1\ score = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} \tag{6}$$

$$False\ negative\ rate = \frac{FN}{TP + FN} \tag{7}$$

where TP = true positives; TN = true negatives, FP= false positives, and FN = false negatives.

7.1 Hyperparameter tuning

To increase the model performance, hyperparameter tuning was applied using Apache Spark's implementation of the grid method. This method requires that all values for a given hyperparameter are set previously; the method then tests the performance of every possible combination of the given hyperparameter values and returns the model that offers the best performance in a predetermined performance metric. The values tested for each hyperparameter are presented in Table 2, with Spark's default values highlighted in bold. For the multilayer perceptron, four different topologies for the hidden layers were evaluated: a single layer (i) with 18 neurons, recommended value of neurons equal to the average between the number of features and the number of labels, and (ii) with 70 neurons, to evaluate a single layer with a greater number of neurons; (iii) two layers with respectively 10 and 8 neurons, and (iv) three layers with 6 neurons in each layer, to assess the effect of increasing the number of layers. As there is no default value for this hyperparameter, the topology containing a single layer with 18 neurons was considered as the default.

Table 2 Algorithms tested and their hyperparameter values used on grid optimization

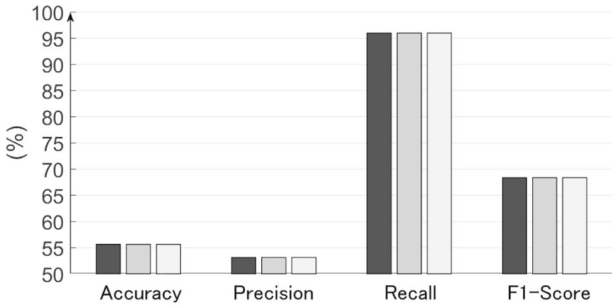
Naïve Bayes	
Smoothing	[0.0, 0.25, 0.5, 0.75,**1.0**, 2.5, 5.0]
Logistic regression	
ElasticNet param	[**0.0**, 0.25, 0.5, 0.75, 1.0]
Max iterations	[5, 10, 20, 50, **100**, 200]
Regularization param	[**0.0**, 0.01, 0.1, 0.3]
Tolerance	[$\mathbf{10^{-6}}$, 10^{-5}, 10^{-4}, 10^{-3}]
Support vector machine	
Fit intercept	[**True**, False]
Max iterations	[5, 10, 20, 50, **100**, 200]
Regularization param	[**0.0**, 0.01, 0.1, 0.3]
Standardization	[**True**, False]
Tolerance	[$\mathbf{10^{-6}}$, 10^{-5}, 10^{-4}, 10^{-3}]
Multilayer perceptron	
Hidden layer topology	[(18), (70), (10,8), (6,6,6)]
Max iterations	[5, 10, 20, 50, **100**, 200]
Step size	[0.001, 0.01, **0.03**, 0.1]
Tolerance	[$\mathbf{10^{-6}}$, 10^{-5}, 10^{-4}, 10^{-3}]
Decision tree	
Impurity	[**"Gini"**, "Entropy"]
Max depth	[3,**5**, 6, 9, 12, 15, 18, 21, 24, 27, 30]
Min info gain	[**0.0**, 0.1, 0.2, 0.3]
Min instances per node	[**1**, 2, 5, 10, 20, 40]
Random forest	
Max depth	[3, **5**, 6, 9, 12, 15, 18, 21, 24, 27, 30]
Number of trees	[**20**, 50, 100, 200, 300, 400, 500]
Subsampling rate	[0.5, 0.75, **1.0**]
Gradient-boosted tree	
Max depth	[3, **5**, 6,9, 12, 15, 18, 21, 24, 27, 30]
Step size	[0.05, **0.1**, 0.2]
Subsampling rate	[0.5, 0.75, **1.0**]

The target metrics chosen were the precision, to minimize the false-positive rate, and F1-score, to obtain a balance between both false positives and false negatives. Figure 3 contains the obtained results for the evaluated algorithms. FNR results were presented only for the decision tree, random forest, and gradient-boosted tree, which offered the best performance results in classification considering both precision and F1-score. The results show the average value obtained for each test; the tests consider a confidence interval of 95%; however, these intervals were omitted since they are not significant considering the graphs' scale.

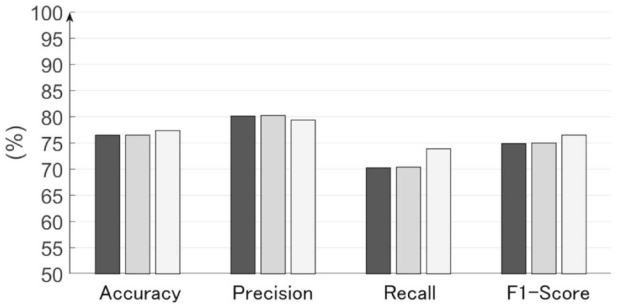
Figure 3a presents the results of hyperparametric optimization for naive Bayes. The optimal results for all tested cases, both with and without optimization, were the same. This occurred because the default value offered by the platform for the adjusted hyperparameter, smoothing, already offers the best results for the dataset used. As can be seen in Fig. 3a, the model has high sensitivity, but the precision results are close to 50%. This indicates that the model has a high rate of false positives, which for an application aimed at detecting network threats is catastrophic, as it results in almost half of the normal flows being erroneously classified as malicious traffic. This demonstrates the importance of analyzing multiple metrics in building a model, as well as why F1-score is considered a more complete metric than just accuracy or precision.

Figure 3b presents the performance results obtained for logistic regression. In the case of optimization for precision, the optimal hyperparameters follow the standard values, with the only difference being a convergence tolerance of 10^{-3}. By acting as a stopping criterion, lower tolerance of convergence potentially results in a model with less overfitting, slightly improving the classification performance. Optimization for F1-score, on the other hand, offered a more significant result, using 10 maximum iterations, a regularization parameter 0.3, and 10^{-3} convergence tolerance instead of the default values of 100, 0.0, and 10^{-6}, respectively. Using a smaller number of iterations offers a less accurate model, but the greater sensitivity and lesser overfitting resulting from changes in the other two hyperparameters compensate for this loss in the calculation of the F1-score.

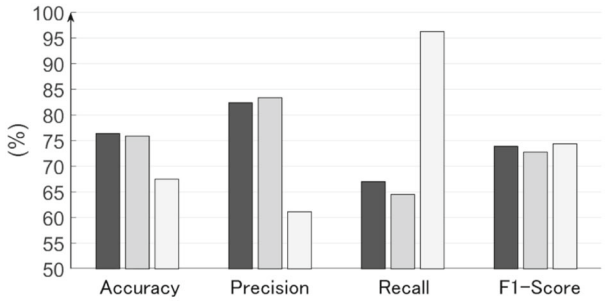
Figure 3c shows the performance results obtained for SVM. Similar to logistic regression, the adjusted



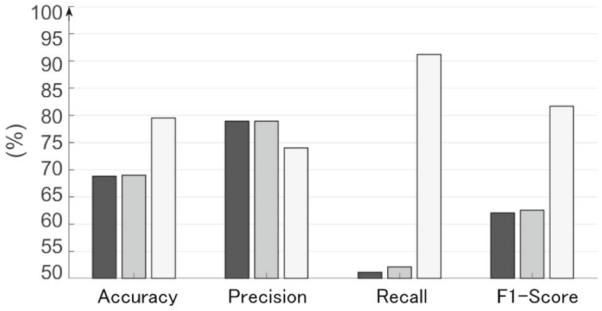
(a) Naive Bayes (NB).



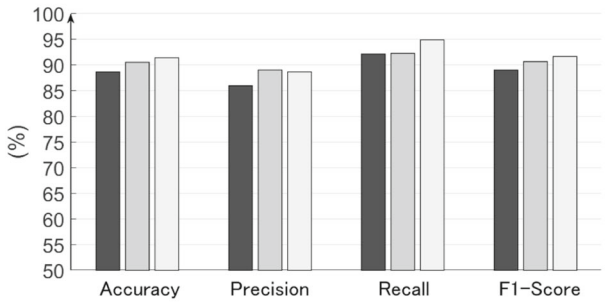
(b) Logistic Regression (LR).



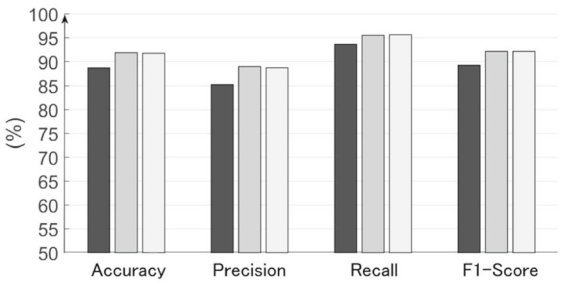
(c) Support Vector Machine (SVM).



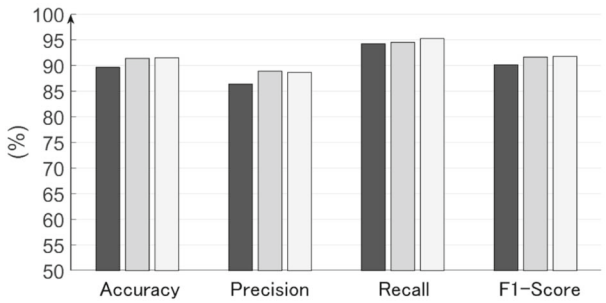
(d) Multilayer Perceptron (MLP).



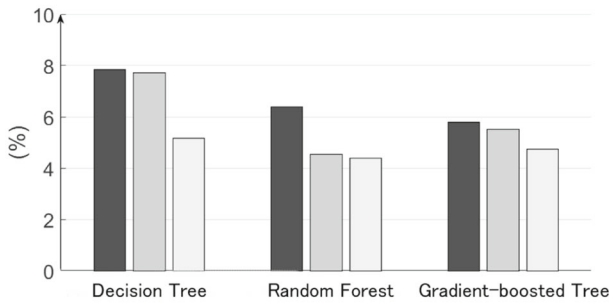
(e) Decision Tree (DT).



(f) Random Forest (RF).



(g) Gradient-Boosted Tree (GBT).



(h) False Negative Rates (FNR).

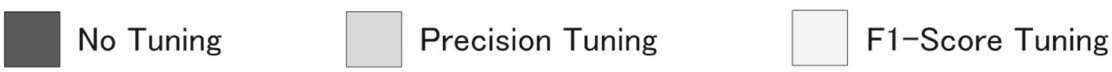


Fig. 3 (a)–(h) Performance of the metrics of accuracy, precision, sensitivity, F1-Score and FNR for the evaluated classifiers, for cases without optimization and with optimization of precision and F1-Score as target metrics. The FNR is only calculated for the classifiers with satisfactory performance in the precision and F1-Score metrics

hyperparameters seek to define a better stop condition for the algorithm. In optimizing for precision, the optimal model reduced the maximum number of iterations to 50, in addition to also reducing the convergence tolerance to 10^{-3}, both modifications to reduce overfitting. In the optimization for F1-score, the maximum number of iterations and the convergence tolerance has also been reduced to 10 and 10^{-3}, respectively. Regularization was also applied, with the regularization parameter set to 0.3.

Figure 3d presents the results obtained after optimizing the multilayer perceptron. In the case of optimization with precision as target metric, the best result applied a single hidden layer with 18 neurons, in addition to increasing the maximum amount of iterations from 100 to 200 and reducing the step size from 0.03 to 0.001. While tuning for F1-score, the topology that provided the optimal result was composed of 3 hidden layers with 6 neurons in each layer. There was also an increase in the maximum number of iterations from 100 to 200, as well as a reduction in the convergence tolerance from 10^{-6} to 10^{-4}. The results demonstrate the importance of topology in the performance of the multilayer perceptron, with the topology containing more layers offering a significantly lower false negative rate than the model containing a single layer. This algorithm was the most impacted by the F1-score optimization, having a percentage gain in FNR greater than 83%; however, the final performance of the model was still inferior to the decision tree-based algorithms before the hyperparemeter tuning.

Figure 3e presents the performance results for the decision tree. When optimizing for precision, the only hyperparameter with a value other than the default was the maximum depth, which went from 5 to 30. In F1-score optimization, the maximum depth went from 5 to 18, and the method used to calculate the impurity was entropy, instead of the Gini method. Both results demonstrate the importance of the maximum depth hyperparameter, with the tree provided by default by the platform not offering a satisfactory performance. The model optimized for F1-score also offered the greatest gain in FNR among the tree-based algorithms when compared to cases without optimization, as shown in Fig. 3h; this model presented a percentage gain of 34.18%.

Figure 3f presents the optimization results obtained for the random forest algorithm. For optimization with precision as a target metric, the optimal values of hyperparameters were: maximum depth 30, number of trees 50, and subsampling rate of 0.75. For the optimization aiming at F1-score, these values were maximum depth 24 and number of trees 300. In both cases, the values adjusted for maximum depth and number of trees are higher than the standard values offered by Apache Spark. Figure 3h demonstrates how this algorithm offers the lowest FNR after hyperparametric optimization; this reduction was accompanied by the third-highest percentage gain: 31.46% when compared to the case without optimization.

Figure 3g shows the results of the gradient-boosted tree. The hyperparameters that offered the best results were, for optimization with precision as a target metric: maximum depth 18, step size equal to 0.2, and subsampling rate equal to 0.75. For the optimization aiming at F1-score, the only hyperparameter with a final value different from the standard values was the maximum depth, which went from 5 to 15. This reinforces the importance of the depth of the tree in the performance of this family of algorithms, being the only hyperparameter modified to result in the model with the highest performance in the selected metric in all tree-based algorithms. While the random forest shows the best performance after optimization, the gradient-boosted tree shows the best performance using the standard values of hyperparameters, considering both FNR and F1-score.

7.2 Results and analysis

The model convergence and training time plus the processing speed must be considered in the context of real-time analysis. To verify the impact of the data structure used during model training, we compared the model training time of Dataframe-based TeMIA-NT with the RDD-based CATRACA, and IDS previously proposed by our research group. The algorithm used for this comparison was the decision tree since it's the only method implemented by CATRACA. Figure 4 shows that the Dataset data structure's several performance optimizations have a significant impact

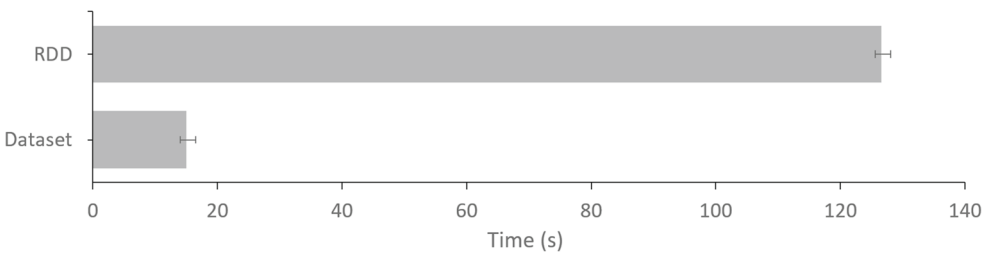


Fig. 4 Impact of the data structure on the training time of the decision tree

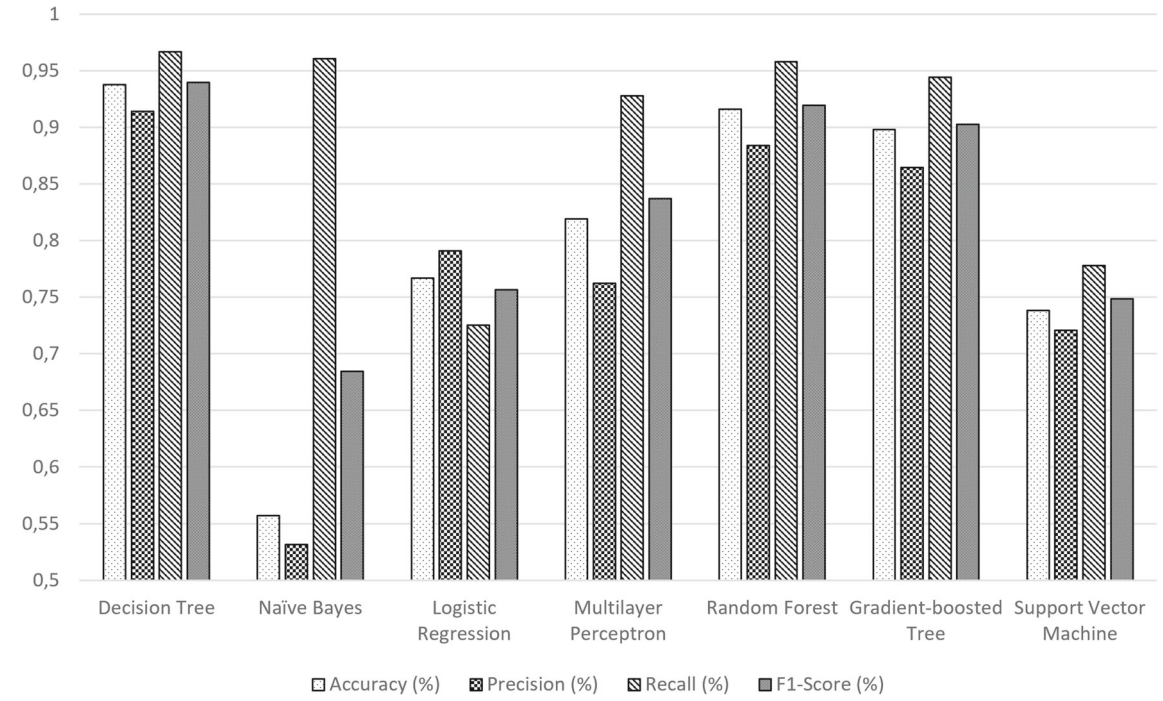


Fig. 5 Comparison of the evaluation metrics for the seven classifiers

on the latency. Training the model with DataFrame is eight times faster than the same operation made with RDD.

We split the dataset into 70% for the training set and the other 30% for the test set to obtain the models in the offline processing mode. Also, we used K-fold cross-validation, with $k = 10$, to guarantee the model's generalizability. Finally, we use the grid search method to tune the hyperparameters in each algorithm. Figure 5 shows the results of each algorithm with weighted F1-score set as the target classification metric. Similar to the results presented in Figs. 3 and 5 presents the average value obtained for each algorithm, with the confidence interval of 95% omitted since it is not significant considering the graph's scale. Based on Fig. 5, random forest, decision tree, and gradient-boosted tree models offer the best performance.

The online mode of operation uses the model with the highest processing capacity and good accuracy. Table 3 shows that the decision tree algorithm presented the maximum flow volume rate of 50 GB/s. The random forest classification model has a lower performance due to the need to process multiple trees, and it is necessary to obtain the result for all trees to achieve the final classification result.

Table 3 Models processing efficiency with the best results in terms of the number and volume of classified flows per second

	Flows/s	GB/s
Random forest	586.563,32	21,95
Decision tree	1.330.732,59	49,80
Gradient-boosted tree	1.206.962,94	45,17

As the decision tree model presents the best results both in accuracy and in classification capacity, this is the model used by default in the execution of the proposed system. However, other models can also be used according to the user's needs.

The last test observed the impact of parallelism on the system's performance, observing how variations in the number of processing nodes affect the results. The processing time required to classify 10 million network flows was measured while varying the number of processing nodes between 1 and 4 in the Spark environment; the results can be seen in Fig. 6.

As can be seen from Fig. 6, increasing the number of processing nodes reduces the total time required to process flows for most algorithms. This impact is more significant in the case of the random forest, as this algorithm works by creating multiple models of trees that can be executed in parallel during the classification process. The results of the decision tree algorithm are negatively affected by the increase in the number of nodes.

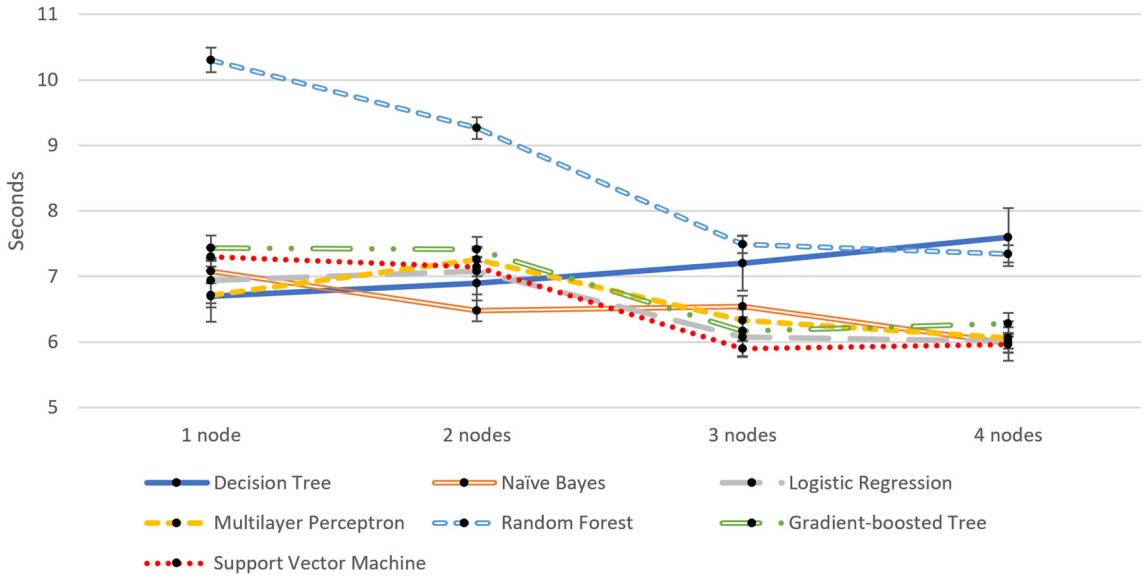


Fig. 6 Time necessary for classification based on number of processing nodes

8 Conclusion

This article presents the TeMIA-NT system,[3] developed to monitor traffic using parallel flow processing. TeMIA-NT presents two modes of operation: online and offline. The online mode of operation allows the network manager to monitor and detect network security threats in real-time. The offline mode of operation allows the performance evaluation of multiple classification models obtained from different algorithms and datasets. TeMIA-NT also allows the selection from seven machine learning algorithms when obtaining classification models. The detection of threats in real time with low latency is achieved thanks to the dataframe data structure and the continuous processing engine of the structured streaming library.

The obtained results from a dataset based on legitimate traffic demonstrate the high processing capacity in flows per second. The performance of each implemented machine learning algorithm is also observed, with the decision tree and random forest models presenting high values in metrics such as accuracy and F1-score. The results also demonstrate the positive impact of hyperparametric optimization on the performance of algorithms, aiming to minimize the amount of incorrectly classified network attacks. It was observed how the multilayer perceptron model was the most affected by the tuning, due to the hyperparameter that defines the hidden layers topology. A percentage gain of 83.1% was observed in the false negative rate (FNR) when comparing the best performance obtained to the default hyperparameter values. It was also observed that the best models in the detection of network threats, the decision tree, and random forest, presented a gain greater than 30% in FNR when compared to cases without optimization. This reduction was obtained while maintaining good accuracy and F1-Score, demonstrating the beneficial impact of making the optimization even in models that originally presented good results. It was also shown how most algorithms scale with an increasing number of nodes on an Apache Spark cluster.

[3]The code, documentation, and license are available at: https://www.gta.ufrj.br/TeMIA-NT/.

Funding This work was financed by CNPq, CAPES, FAPERJ, and FAPESP (2018/23292-0, 15/24485-9, 14/50937-1).

References

1. Cybersecurity Market Report. Available at: https://cybersecurityventures.com/. Last access: 30 April 2021
2. Bertino E, Islam N (2017) Botnets and internet of things security. Computer 50(2):76–79
3. Azmoodeh A, Dehghantanha A, Choo K.-K. R. (2019) Big data and internet of things security and forensics: challenges and opportunities. In: Handbook of big data and IoT security. Springer, pp 1–4
4. (2019) Symantec, Internet security threat report. Available at: https://docs.broadcom.com/doc/istr-24-2019-en. Last access: 30 April, 2021
5. Habeeb RAA, Nasaruddin F, Gani A, Hashem IAT, Ahmed E, Imran M (2019) Real-time big data processing for anomaly detection: a survey. Int J Inf Manag 45:289–307
6. (2020) Verizon Enterprise, Data breach investigations report. Available at: https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf. Last access: 30 April 2021
7. Lopez MA, Mattos DMF, Duarte OCMB, Pujolle G (2019) Toward a monitoring and threat detection system based on stream processing as a virtual network function for big data. Concurr Computat Pract Experience 31(20):e5344

8. Pelloso M, Vergutz A, Santos A, Nogueira M (2018) A self-adaptable system for DDoS attack prediction based on the metastability theory. In: 2018 IEEE global communications conference (GLOBECOM), pp 1–6
9. Viegas E, Santin A, Bessani A, Neves N (2019) Bigflow: Real-time and reliable anomaly-based intrusion detection for high-speed networks. Futur Gener Comput Syst 93:473–485
10. Campiolo R, dos Santos LAF, Monteverde WA, Suca EG, Batista DM (2018) Uma arquitetura para detecção de ameaças cibernéticas baseada na análise de grandes volumes de dados. In: Anais do I Workshop de Segurança Cibernética em Dispositivos Conectados. SBC
11. Lobato AGP, Lopez MA, Sanz IJ, Cardenas AA, Duarte OCM, Pujolle G (2018) An adaptive real-time architecture for zero-day threat detection. In: 2018 IEEE international conference on communications (ICC). IEEE, pp 1–6
12. Lopez MA, Mattos DMF, Duarte OCMB (2016) An elastic intrusion detection system for software networks. Ann Telecommun 71(11-12):595–605
13. Lopez MA, Mattos DMF, Duarte OCMB, Pujolle G (2019) A fast unsupervised preprocessing method for network monitoring. Ann Telecommun 74(3-4):139–155
14. Cisco Systems (2014) OpenSOC: The open security operations center. Available at: https://opensoc.github.io/. Last access: 30 April 2021
15. (2017) Apache Software Foundation, Apache Metron. https://metron.apache.org/. Last access: 30 April 2021
16. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: Cluster computing with working sets. HotCloud 10(10-10):95
17. Jirsik T, Cermak M, Tovarnak D, Celeda P (2017) Toward stream-based IP flow analysis. IEEE Commun Mag 55(7):70–76
18. Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ et al (2016) Apache Spark: a unified engine for big data processing. Commun ACM 59(11):56–65
19. Xin R, Rosen J (2015) Project tungsten: bringing apache spark closer to bare metal. Available at: https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html. Last access: 30 April 2021
20. Armbrust M, S Xin R, Lian C, Huai Y, Liu D, K Bradley J, Meng X, Kaftan T, Franklin MJ, Ghodsi A et al (2015) Spark SQL: Relational data processing in Spark. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. ACM, pp 1383–1394
21. Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S et al (2016) Mllib: machine learning in apache spark. J Mach Learn Res 17(1):1235–1241
22. Friedman J, Hastie T, Tibshirani R et al (2001) The elements of statistical learning, vol 1. Springer series in statistics, New York
23. Breiman L (2001) Random forests. Mach Learn 45(1):5–32
24. Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140
25. de Souza LAC et al (2020) DFedForest: Decentralized federated forest. In: 2020 IEEE Blockchain, pp 90–97
26. Chen J, Li K, Tang Z, Bilal K, Yu S, Weng C, Li K (2016) A parallel random forest algorithm for Big Data in a Spark cloud computing environment. IEEE TPDS 28(4):919–933
27. Tavallaee M, Bagheri E, Lu W, Ghorbani AA (2009) A detailed analysis of the KDD CUP 99 data set. In: 2009 IEEE symposium on CISDA, pp 1–6
28. Lopez MA, Silva RS, Alvarenga ID, Rebello GAF, Sanz IJ, Lobato AG, Mattos DMF, Duarte OC, Pujolle G (2017) Collecting and characterizing a real broadband access network traffic dataset. In: 2017 1st cyber security in networking conference (CSNet), pp 1–8
29. Arndt D (2011) Flowtbag. Available at: https://github.com/DanielArndt/flowtbag/wiki/features. Last access: 30 April 2021
30. Reddy T, Boucadair M, Patil P, Mortensen A, Teague N Distributed denial-of-service open threat signaling (dots) signal channel specification, Internet Requests for Comments, RFC Editor, RFC 8782, 05 2020. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8782