IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 13 October 2021; revised 23 May 2022; accepted 24 May 2022.
Date of publication 3 June 2022; date of current version 6 December 2022.

Digital Object Identifier 10.1109/TETC.2022.3178631

On the Properness of Incorporating Binary Classification Machine Learning Algorithms Into Safety-Critical Systems

MOHAMAD GHARIB, TOMMASO ZOPPI, AND ANDREA BONDAVALLI, (Member, IEEE)

The authors are with the Informatic and Maths Department (DiMal), University of Florence, 50121 Firenze, Italy
CORRESPONDING AUTHOR: MOHAMAD GHARIB (mohamad.gharib@ut.ee)

This work was supported by H2020 programme under the Marie Sklodowska-Curie under Grant 823788 (ADVANCE) project, in part by Regione Toscana under Grant POR FESR 2014-2020 SPaCe Project, and in part by European Social Fund via IT Academy Programme.

ABSTRACT Manufacturers are willing to incorporate Machine Learning (ML) algorithms into their systems, especially those considered as Safety-Critical Systems (SCS). ML algorithms that perform binary classification (i.e., Binary Classifiers (BCs)) find a wide applicability as error, intrusion or failure detectors, provided that their performance complies with SCS safety requirements. However, the performance analysis of BCs relies on metrics that were not developed with safety in mind and consequently may not provide meaningful evidence to decide whether to incorporate a BC into a SCS. In this paper, we empirically assess the properness of such incorporation by analyzing the distribution of misclassifications of BCs instead of simply counting misclassifications. This allows us to better assess the adequacy of a given BC by identifying areas of the classification space where the BC is likely to misclassify and therefore constitutes actionable information to deal with the SCS. Our assessment takes a deeper view of the classification performance concerning safety by using new metrics that consider the proportions of predictions that are/are not considered sufficiently safe to be used by incorporating SCS. The results of our experiment allow discussing the potential of such distribution analysis for deciding if a BC can be incorporated into a SCS.

INDEX TERMS Machine learning, performance metrics, safety measures, safety-critical systems

I. INTRODUCTION

Powered by their ability to work with novel input and incomplete knowledge, their learning and generalization capabilities, Machine Learning (ML) algorithms became a highly desirable mean to solve complex problems [1]. Particularly, ML algorithms which perform binary classification (binary classifiers from now on) can efficiently classify input data as belonging either to a positive or to a negative class [8], [10], [19]. Binary classifiers can therefore perform complex tasks such as failure prediction, error detection, intrusion detection, pattern recognition, and even control to mention a few [1]. That is why we are witnessing increased incorporation of binary classifiers into many automated systems covering almost all the main domains of our lives. Particularly, there is a growing interest in integrating binary classifiers into systems whose failure may cause injuries or even death to humans: those systems are classified as Safety-Critical Systems (SCS) [2]. In SCSs, the adoption of binary classifiers needs to be regulated by rigorous safety requirements as such systems require levels of assurance far beyond those needed for non-SCS. More specifically, the performance of a binary classifier must be assessed and guaranteed to be compliant with safety requirements of incorporating SCS before it is used in its operational environment [3].

For instance, modern automotive vehicles have been proven vulnerable to hacking attacks by exploiting vulnerabilities in their external interfaces [4], through which, an attacker can access the Controller Area Network (CAN) bus and control some core functions of the vehicle. An example of such attacks is the hijacking of the steering and braking units in a Ford Escape[1]. Similarly, a group of hackers was able to remotely hijack a Tesla Model S from a distance of

1.Greenberg, A., 2013. Hackers Reveal Nasty New Car Attacks-With Me Behind The Wheel (Video). https://bit.ly/3lWRAIN

around 12 miles[2]. Moreover, Chrysler announced a recall for 1.4 million vehicles after a pair of hackers demonstrated that they could remotely hijack a Jeep's digital systems over the Internet[3]. Overall, adverse safety impacts of such attacks can be prevented or mitigated by promptly detecting any malicious/anomalous behavior on the CAN bus [4], which can be done relying on binary classifiers that detect intrusions. Intrusion detectors should then trigger adequate reaction strategies, e.g., switching off connectivity, blocking specific network addresses. Should the intrusion detector fail to identify such malicious/anomalous behavior (i.e., a False Negative, FN), the attacker can carry on its attack controlling the vehicle that might cause tragic incidents.

The vast majority of existing metrics for assessing the performance of binary classifiers mainly focus on some properties of interest in the domains where they were developed [5]. Many researchers have advocated that metrics should be domain-specific, yet very few efforts have been made to propose more safety-oriented metrics that focus specifically on FN predictions and their distribution. For example, Accuracy is one of the most commonly used metrics that provides an overall performance evaluation of binary classifiers focusing on the number of correct predictions i.e., True Positive (TP) and True Negative (TN). Other metrics such as Precision and F-score have been designed to focus mainly on the number of TP predictions [5], and they have less emphasis on False Negative (FN) predictions. To this end, existing methodologies to evaluate the properness of integrating binary classifiers into SCSs that do not explicitly focus on FNs and on how they are produced may not be appropriate for assessing the performance of binary classifiers with respect to the safety requirements of the incorporating SCS.

In a previous work [6], we worked toward solving this problem by proposing a theoretical technique to evaluate binary classifiers by isolating predictions that are sufficiently guaranteed to be correct from other predictions based on distribution of scores of the binary classifier itself. However, we did not extensively motivate, nor implement, evaluate and provide clear indications on how to practically take advantage of distribution analysis.

Particularly, this paper shows the importance of considering how FNs are distributed, which is very important in deciding whether a prediction of a binary classifier can or cannot be considered sufficiently safe to be used by a SCS. Here, we elaborate and expand on:

- A discussion on a fail-controlled architecture to integrate binary classifiers into SCSs.
- Motivation on why analyzing the distribution of scores and predictions of binary classifiers rather than just counting misclassifications clearly helps in assessing the properness of incorporating binary classifiers into SCSs.

2.Solon, O., 2016. Team of hackers take remote control of Tesla Model S from 12 miles away | Technology | The Guardian. https://bit.ly/3h73FaL
3.Greenberg, A., 2016. The Jeep Hackers Are Back to Prove Car Hacking Can Get Much Worse. https://bit.ly/35e9CAk

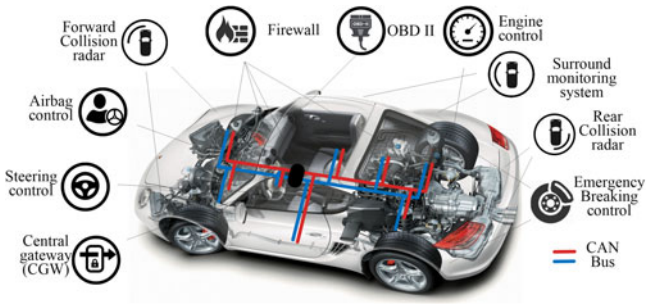

FIGURE 1. A representation of the architecture of vehicle with a CAN Bus.

- Propose and implement a technique to separate the predictions of binary classifiers that are sufficiently safe to be used by incorporating SCS from other predictions, and quantitative metrics to support an "informed decision" on whether a binary classifier can be incorporated in SCS.
- Empirical assessment of our solution by exercising 12 binary classifiers on 9 public datasets related to SCSs. Such assessment does not aim at comparing the performance of binary classifiers: instead, it shows how to calculate the sufficiently safe predictions of a binary classifier for a specific system, and how they help in deciding if a binary classifier should be used in a SCS.
- Report on our findings and lessons learned while assessing the properness of incorporating binary classifiers into SCS through the aforementioned experiment.

The rest of the paper is organized as follows; Section II describes a motivating example to illustrate the relevance of this work. Section III presents a general fail-controlled architecture for incorporating binary classifiers into SCS. Section IV discusses a way to analyze safety of a binary classifier based on distribution of its scores, which paves the way to formalize distribution-based metrics in Section V. Our experimental methodology is presented and described in Section VI. Section VII discusses the experimental results about the properness of incorporating binary classifiers into a SCS. Section VIII discusses the related works, and Section IX lists threats to the validity of our work, concluding remarks and future works.

II. MOTIVATING EXAMPLE: AN INTRUSION DETECTION SYSTEM FOR CONTROLLER AREA NETWORK (CAN) BUS

There has been significant growth in the number and complexity of electronic components/units in modern vehicles, which may have as many as 70 Electric Control Units (ECUs) [7]. ECUs control various functions ranging from temperature control to steering and cruise control [8]. Some of these units need to communicate with one another, and the CAN bus was developed to prevent the need for large multi-core wiring in modern vehicles. Although the CAN bus is currently the most widely used bus in modern vehicles [8], it lacks appropriate security mechanisms, and it has been proven to be subject to various remote attacks. For instance, an attacker can gain access to the

CAN bus by hijacking one of the connected ECUs [7] since some ECUs communicate with the outside world. Then, the attacker can inject messages into the bus that can invoke sudden braking, turn the engine off, control the steering wheel or the braking system to mention a few.

This problem became more prominent as vehicles are getting more and more connected by technologies such as Bluetooth, WiFi and smart-phones, or as in intelligence connected vehicles, where various units and systems are connected to the CAN bus. In this context, vehicle security became a main concern for the general audience since most people own or use vehicles. Accordingly, they might be subject to such attacks [7]. More specifically, vehicle security is not only related to data confidentiality and integrity anymore, but also to traffic safety that is directly related to humans' safety [9].

A modern vehicle architecture (illustrated in Figure 1) includes a central gateway (CGW), which is connected to the On-board diagnostics (OBD) II port as well as the CAN bus that is connected to various ECUs. Most ECUs can be equipped with firewalls that filter packets and allow white-listed packets to pass only. As monitoring the traffic on the CAN bus allows detecting attacks/suspicious behavior, it is possible to employ a binary classifier as an Intrusion Detection System (IDS) [10]. Such IDS can be installed at least in the CGW but in order to improve the detection rate, it can be additionally deployed in a specific or even all ECUs. Such IDS monitors communication packets and also the ECU status if it is installed in one. Then, it compares such behavior with a reference (intended/specified) behavior to detect abnormal/anomalous behavior, and triggers appropriate security mechanisms [11]. Moreover, to identify both known and unknown (e.g., zero-day attacks [10]) intrusion attacks, the use of unsupervised binary classifiers is advised [10].

In what follows, we specify the *attack model* we considered in this study concerning the IDS example in terms of attackers' capabilities and knowledge.

Capability: Our study assumes an attacker can exploit a vulnerability in one of the external interfaces of the vehicle, and also can inject harmful messages into the CAN bus through the aforementioned interface to compromise an ECU that is responsible for a critical functionality (e.g., braking system, engine, steering wheel). We assume that the attacker does not have access to the training set(s) and does not have the capability to perform data poisoning attacks (e.g., contaminates the training set(s)), which can dramatically degrade the performance of the binary classifier for detecting its planned attacks.

Knowledge: Our study assumes that the attacker is aware of vulnerabilities in at least one of the external interfaces of the vehicle, which can be exploited. The attacker is also capable of exploiting this vulnerability by adequate technical means, and can also inject harmful messages into the CAN bus. Moreover, the attacker should have the basic knowledge of the architecture vehicle in terms of its CAN bus and how it is connected to the targeted ECU.

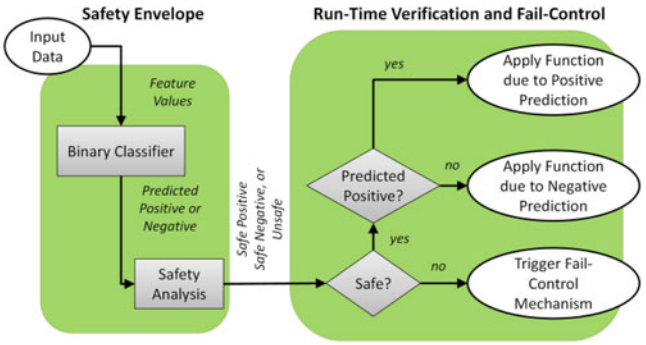

FIGURE 2. A fail-controlled architecture to handle safe and unsafe predictions of a binary classifier.

III. A FAIL-CONTROLLED ARCHITECTURE FOR INCORPORATING BINARY CLASSIFIERS INTO SCSS

Generally speaking, safety management can be described as practices that direct, monitor and intervene in core operations to achieve or maintain safety through fail-aware, fail-safe, or fail-operational mechanisms. These same mechanisms can be used to deal with situations when a binary classifier incorporated into SCS does not work as intended, and therefore may lead to a failure at the incorporating SCS level.

A. SAFETY ENVELOPES

According to Avizienis *et al.* [9], fail-controlled systems are systems designed so to fail only in those modes that are prescribed by their specification and only to an extent considered acceptable. In our work, identifying when the system may fail and providing techniques to deal with such failure is essential to develop a fail-controlled binary classifier. Moreover, Salay and Czarnecki [12] listed two architectural techniques for fault tolerance and discussed how they can be used to deal with faults resulting from binary classifiers:

- *Safety envelope* [13], where a safety component runs alongside with the binary classifier with the main objective of monitoring, assessing and optimizing the behavior of the last concerning the desired safety requirements. More specifically, the monitoring component will assess the predictions of the binary classifier and decide whether such predictions are sufficiently safe to be used by the incorporating system.
- *Runtime verification and fail-safe*, which i) conducts error checking to identify whether some preconditions concerning the predictions of the binary classifier are violated. In such cases, the output cannot be trusted; and ii) employs adequate fail-safe architecture technique that disables the functionality on error and transitions the system to a safe state.

A general-purpose fail-controlled architecture for incorporating binary classifiers into SCS is shown in Figure 2. Such architecture adopts a Safety Envelope to mask the predictions of a binary classifier component into safe and unsafe predictions. Further, instead of adopting a run-time verification and fail-safe principle, the architecture adopts a run-time verification and

fail-controlled principle in order to cover a wider range of mechanisms than just the fail-safe ones, i.e., fail soft recovery [14]. According to this architecture, a binary classifier either i) provides predictions that are sufficiently safe to be used in a SCS, either positive or negative, or ii) triggers fail-control mechanisms to mask a potential ongoing failure and guarantee "safety" in such situations.

B. ACCEPTABLE LEVEL OF RISK AND STANDARDS

Safety does not assume that critical events will never happen in a system. Instead, it guarantees that the risk – a combination of likelihood and impact – of a given threat is tolerable according to the requirements of a given system. In other words, a few misclassifications of a binary classifier might appear as safe predictions. Misclassifications may either be FNs (positive predicted as negative) or FPs (negative predicted as positive): however, they do not have the same impact on most SCS. Whereas FPs might indirectly lead to unsafe situations, in SCSs FNs are generally considered the primary and direct trigger to catastrophic incidents e.g., pedestrian not detected in autonomous driving, undetected attack in the security domain, or component/resource/device misbehavior that has an erroneous behavior in general. We assume here that only FNs may be the direct cause of safety issues, whereas FPs mostly impact the availability and usability properties of a given system or component.

Therefore, we foresee that a given amount of FNs, which we call FN^* (or residual FNs), may happen even in a SCS. However, those FN^* should not exceed the Acceptable Levels of Risk (ALR) corresponding to the safety requirements of the system or component [15]. ALR is a commonly used concept in safety standards (e.g., IEC 61508 [15], CENELEC - EN 50129 [16], ISO 26262:2011 [17]) to specify the tolerable hazard. The ALR can be seen as an indicator for identifying the Safety Integrity Level (SIL), which is a measure of the confidence with which the system is expected to deliver a safety function. Moreover, safety standards such as IEC 61508 [15] address ALR considering two operating modes:

- low demand mode of operation that is dedicated for systems or functions that operate on demand, and this mode is specified in terms of the Probability of Failure on Demand (PFD); and
- the high demand or continuous mode of operation that is dedicated for systems or functions that operate continuously, and this mode is specified in terms of the Tolerable Hazard Rate (THR).

For each of these operating modes, different ALR in terms of THR or PFD are considered.

Noteworthy, existing safety standards (e.g., IEC 61508 [15], ISO 26262:2011 [17]) did not evolve to cover ML algorithms, and they do not provide compliance requirements for binary classifiers nor error handling mechanisms that are explicitly tailored for them. Moreover, most standards define the SIL, THR or PFD for a system function, not for a component. Therefore, specifying the THR/PFD for a binary classifier is not an easy task as it relies on the architecture of the incorporating system: the THR/PFD of the incorporating system can be further apportioned at the level of the binary classifier [16]. To such extent, several researchers relied on safety/assurance cases [18] to derive the safety requirements at the binary classifier level from the incorporating system requirements.

In this paper, we followed the same approach, constructing a safety case for the binary classifier to derive its SIL level as well as corresponding ALR based on the requirements and architecture of the incorporating system. Due to space limitations, we briefly summarize the safety case we considered for determining the ALR for the an IDS that uses a binary classifier. The top-level Goal G1 in our study was "the performance of the IDS for the ECU is compliant with the safety requirement of the incorporating system", which was decomposed into two sub-goals: G2 and G3 that aim at specifying the safety requirement for the incorporating system and the IDS respectively. G2 was achieved by identifying hazards involved, which we performed relying on Hazard Analysis and Risk Assessment (HARA). Then, we conducted a risk assessment for each of the identified hazards following ISO 26262 [17]. Followed by identifying the corresponding ASIL level(s) appropriate to the identified hazards. As a result, ASIL A was determined. On the other hand, G3 was achieved by identifying the ALR of the IDS considering the ASIL of the incorporating system (ASIL A) as well as the operation mode of the IDS (low demand mode). As a result, the ALR of the IDS and of the binary classifier was determined as 0.01[4].

IV. SAFETY ANALYSIS OF A BINARY CLASSIFIER

A. ALGORITHM SCORES AND DECISION FUNCTIONS

Binary classifiers devise a mathematical model from a training data set, which contains a given amount of data points, where each data point contains *f* feature values. Once training is completed, a binary classifier *bc* takes advantage of that model to make predictions concerning new data points [19] through a function:

$$dp\_label = bc.predict(dp)$$

where *dp* is a single data point composed by *f* feature values, and *dp_label* is a binary label – either positive or negative – which represents the (binary) prediction for a data point. Noteworthy, the *predict* function can be broken down as a composition of two different sub-functions

$$bc.predict(dp)\text{: } bc.decisionfunction.apply(bc.score(dp))$$

In particular, a binary classifier assigns numeric scores to each data point *dp* and thus $bc.score(dp)\ \varepsilon\ \mathbb{R}$. For clustering algorithms, this score can be the distance with respect to the centroid of the closest cluster, whereas in distance-based algorithms such numeric score can be derived from a kNN graph [20]. Then, a decision function is applied to such

4.Please refer to Appendix A for a full description of how we constructed a safety case for the binary classifier to derive its SIL level as well as corresponding ALR based on the requirements and architecture of the incorporating system.

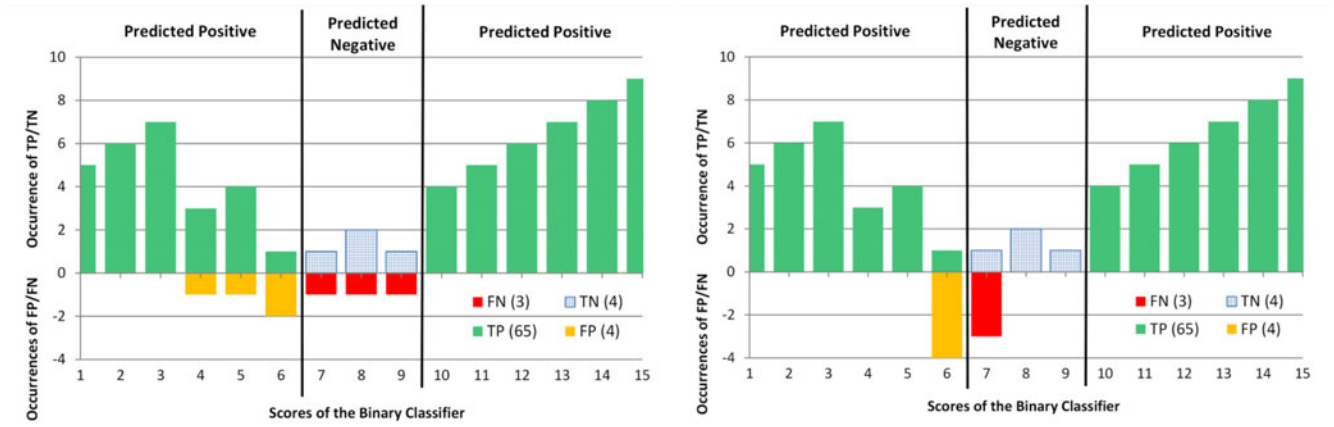

FIGURE 3. (a-left, b-right): Different distribution of the same amount of misclassifications. X-axis reports on the score of the binary classifier, whereas the y-axis reports the amount of correct classifications (above the horizontal line) or misclassifications (below the horizontal line). Decision Function (Negative if $7 \leq$ num_score ≤ 9) are represented as vertical lines and show the binary classification result.

numeric score *num_score* to convert it into classes; in binary classification, a decision function converts a numeric score into a binary label [21], either positive or negative.

bc.decisionfunction.apply(num_score) ε *{positive, negative}*

A decision function identifies one or more classification thresholds, e.g., interquartile range [22] or confidence interval. Depending on the score of a data point, a classification decision is made: data points with scores higher than the threshold are predicted to belong to one class, whereas data points with scores lower than the threshold are predicted to belong to the other class. Decision functions such as interquartile range, confidence interval and even majority voting are commonly used, albeit some binary classifiers require algorithm-specific thresholds. Supervised binary classifiers as the k-th nearest neighbor [20] calculate the *score* depending on the amount of those k neighbors that have a given label. Then, this numeric amount is converted into a binary label through majority voting (i.e., is the numeric score bigger than k / 2). Instead, the unsupervised version of kNN [20], calculates the (Euclidean) distance of a data point with respect to its k-th neighbor, and then applies a decision function based on a single threshold (e.g., if the distance is bigger than 100, a data point is predicted positive) to derive the binary label.

As such, predictions of a binary classifier can be partitioned into four mutually exclusive groups based on the real class of the data point – the *ground truth* - and the class predicted by the binary classifier: True Positive (TP) cases refer to the Predicted Positives that are indeed positives; True Negatives (TN) cases refer to the Predicted Negatives that are indeed negatives; False Positive (FP) cases refer to the Predicted Positives that are instead negatives; and False Negatives (FN) cases refer to the Predicted Negatives that are positives. Altogether, those four classes build the so-called confusion matrix, which is commonly used to evaluate the classification performance of binary classifiers. In fact, commonly used performance metrics [5] aggregate items of the confusion matrix to build Precision, Recall (or Coverage), False Positive Rate (FPR), Accuracy (ACC), FScore-β (Fβ), F-Measure (F1), Area Under ROC Curve (AUC) and Matthews Coefficient (MCC).

B. THE DISTRIBUTION OF MISCLASSIFICATIONS

However, metrics based on the confusion matrix may not adequately describe all the aspects of the behavior of a binary classifier. Suppose that a binary classifier *bc1* was trained using a given training set. Such algorithm provides numeric scores (*bc.scores(dp)* function) in the range of [1];[15] for a given data point. Additionally, the algorithm decides on binary classification by applying the following decision function: negative if $7 \leq num\_score \leq 9$, and positive otherwise. Now, we provide the binary classifier with 76 data points that build our test set. The application of *bc1* results in 65 TP, 4 TN, 4 FP and 3 FN, which translate to values of 90.7 and 94.9 of accuracy and F-Measure (or F1-Score). Figure 3a plots the frequency of each of the four classes of the confusion matrix against a numeric score *num_score* of the algorithm through a bar chart where bars above the x-axis show TP or TN, and bars below x-axis depict FP or FN. It is worth noticing how misclassifications – either FN or FP – happen when the binary classifier assigns a numeric score in the range [4]; [9]. Now, let's consider a different binary classifier *bc2* trained on the same training set which shows the same amount of misclassifications on the test set, but with a slightly different distribution as shown in Figure 3b. The confusion matrix is the same for *bc1* and *bc2* as the number of misclassifications is the same. However, in this picture, we can observe how misclassifications by *bc2* accumulate only on scores in the range [6]; [7].

This difference in the behavior of the two binary classifiers is not reflected in the confusion matrix: misclassifications have different distributions. As a consequence, *considering performance metrics uniquely tied to the amount of TP, TN, FP, and FN, not accounting for their distribution against scores* may hide some details of the behavior of a binary

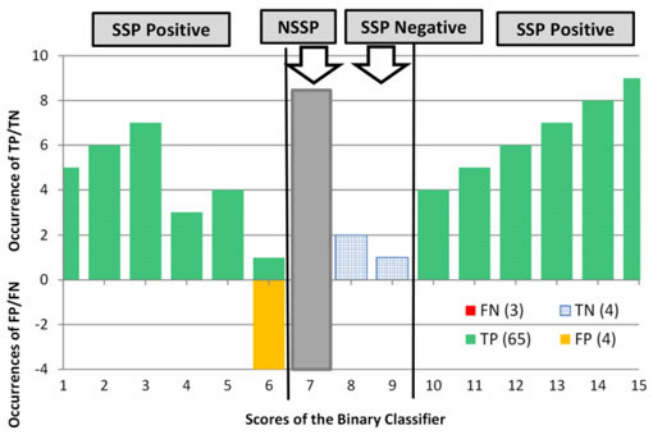

FIGURE 4. **Semi-ternary Classification into SSP (either positive or negative) and NSSP from example in Figure 3b.**

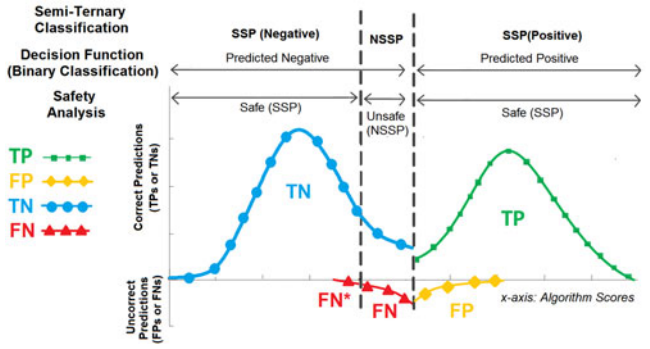

FIGURE 5. **Semi-ternary Classification into SSP (either positive or negative) and NSSP.**

classifier. These details may turn out to be critical when deciding on incorporating a binary classifier into a SCS.

C. FROM BINARY TO SEMI-TERNARY CLASSIFICATION

Here comes the intuition behind the rest of the paper.

It is possible to analyze the distribution of misclassifications to identify which numeric scores assigned by the binary classifier may generate misclassifications, and especially FNs, after applying the decision function. Accordingly, we will consider this subset of scores as *not sufficiently safe*, because we know that when the binary classifier assigns those scores it can potentially output a FN. Therefore, we depict a gray area (see Figure 4) in which predictions are not sufficiently safe (NSSP), while in the rest of the scores the algorithm provides predictions that are sufficiently safe (SSP), either positive or negative. In the example in Figure 4, we will have 69 SSP Positive, 3 SSP Negative, and 4 NSSP. Overall, out of the 76 data points in the test set, we will have 72 predictions that are safe, and 4 predictions that we know are not safe to be used in SCSs.

In our previous work [6], we observed that a great number of misclassifications (FN and/or FP) usually co-locate around the threshold(s) of the decision function. Therefore, we proposed a technique to isolate predictions that are sufficiently guaranteed to be safe (TP, TN, and, to a lesser extent, FP) from other predictions (FN and TNs that co-locate with them) relying on two thresholds. As such, any prediction which falls outside the two thresholds is considered sufficiently guaranteed to be safe and can be used by an incorporating SCS.

To generalize this formulation, we show in Figure 5 a general representation of the distribution of algorithm scores according to the confusion matrix, separating correct predictions (TNs and TPs) from incorrect (FPs or FNs) ones, which are below the x-axis. The figure shows two vertical dashed lines which contain the Not Sufficiently Safe (NSSP) area, whereas items outside those thresholds are to be considered as Sufficiently Safe Predictions (SSP). Noticeably, not all FNs are inside the NSSP: there is a slight percentage, labelled as FN^*, which appear as SSP even if they have detrimental impact on safety. This is the portion of FNs that is deemed tolerable according to a given ALR: therefore, $FN^* \leq ALR$. Overall, predictions of a binary classifier are said to be sufficiently safe to be used (SSP) by incorporating SCS (in which FN are the problem) when:

- They are predicted positive, either TP or FP. Noticeably, a FP is an incorrect prediction that may not lead to unsafe situations but result in false alarm (e.g., marginal incident).
- They are predicted negative and they do not co-locate to any FN, or co-locate with a residual fraction of FNs such that $FN^* \leq ALR$.

In all other cases, the predictions of the binary classifier are said to be Not Sufficiently Safe (NSSP) and should not be used by the incorporating SCS as they have the potential to originate an unfortunate incident. Ideally, we would like the SSP predictions to be as many as possible, leaving the NSSP area with just a sliver of data points for which the binary classifier cannot deliver a safe prediction.

V. CRITERIA FOR ASSESSING THE PROPERNESS OF INCORPORATING BINARY CLASSIFIERS IN SCS

In this section, we will elaborate on how we can separate the SSP and NSSP from one another, deriving metric scores to quantitatively assess safety of a binary classifier. As explained earlier, the safety of predictions will be defined with respect to an ALR which the safety specialists have to extract from non-functional system requirements. Therefore, in this section we tie the safety of predictions of a binary classifier to a parameter *ALR*, to derive SSP_{ALR} and $NSSP_{ALR}$ values.

A. AN ALGORITHM TO SEPARATE SSP FROM NSSP

Let *bc* be the target binary classifier, let V be its validation set with $|V| = nv$ items and let *bcScores* be a set of *nv* numeric scores assigned by the algorithm *bc* to each of the data points in V. Those scores are then converted into either positive or negative prediction according to a decision function *bc.df*. We define the following sets and variables:

Listing 1: Pseudocode to compute SSP_{ALR} and $NSSP_{ALR}$.

```
1 [SSP_ALR, NSSP_ALR]: calcSSP(neg_s, fn_s, ALR, pos)
2   ssp_fn = fn_s
3   [FN*, SSP] = compute(neg_s, fn_s, pos, ALR)
4   while ssp_fn not empty and FN* > ALR do
5     l_fn = ssp_fn.remove(max(ssp_fn.score))
6     r_fn = ssp_fn.remove(min(ssp_fn.score))
7     [l*, lSSP] =
          compute(neg_s, fn_s, l_fn, pos, ALR)
8     [r*, rSSP] =
          compute(neg_s, fn_s, r_fn, pos, ALR)
9    FN* = min(l*, r*)
10   if l* < r* then
11       SSP = lSSP, ssp_fn = l_fn
12    else SSP = rSSP, ssp_fn = r_fn
13   end while
14   return [SSP, pos + count({s_i∈neg_s}) - SSP]

15 [FN*,SSP] :
       compute(neg_s, fn_s, ssp_fn, pos, ALR)
16    FN* = residualFN(neg_s,  ssp_fn, pos)
17    if FN* == 0 then
18      SSP = pos +  count({s_i ∈ neg_s})
19    else if FN* ≤ ALR then
20      SSP = pos + count({s_i∈neg_s |
            s_i < min(ssp_fn.score) or
              s_i > max(ssp_fn.score)})
21    else
22      SSP = pos + count({s_i∈neg_s |
            s_i < min(fn_s.score) or
              s_i > max(fn_s.score)})
23    return [FN*, SSP]

24 FN*: residualFN(neg_s, ssp_fn, pos)
25    if ssp_fn is empty then
26      return 0
27    else return count({s_i∈neg_s |
         min(ssp_fn.score)≤s_i≤max(ssp_fn.score)})
            / (count({s_i∈neg_s}) + pos)
```

- *neg_s*: subset of *bcScores* for which the application of *bc.df* generates a negative prediction. In the example in Figure 3a, neg_s = {7, 7, 8, 8, 8 9, 9} and in Figure 3b neg_s = {7, 7, 7, 7, 8, 8, 9};
- *fn_s*: subset of *neg_s* for which *bc* generates a FN, organized as couples <score, frequency> without repetitions of score values. In the example in Figure 3a, *fn_s* = *{<7,1>, <8,1>, <9,1>}*, whereas in the example in Figure 3b, *fn_s* = *{<7, 3>}*;
- *pos*: the integer amount of scores for which the application of *bc.df* generates a positive prediction. This is always calculated as FP + TP: in the example in Figures 3a and 3b, *pos* = 4 + 65 = 69.

Those variables are used as inputs for the pseudocode shown in Listing 1. Briefly, this greedy algorithm starts (line 3) by calculating the default FN^* residual which - at the beginning – this represents the percentage of FNs over all the *nv* items. In addition, it sets the *ssp_fn* variable as equal to *fn_s*: this describes the FNs that are tolerated in the SSP area. Then, line 4 iterates over the greedy algorithm until the FN^* gets smaller than the ALR, or until the *ssp_fn* becomes empty. Each iteration aims at reducing the FN^* until they get smaller than the ALR. This is done by progressively enlarging the NSSP and thus removing one or more couples from the *ssp_fn* variable, whose size reduces by one item iteration by iteration. The choice on how to enlarge NSSP follows a greedy rule at each iteration: we compute FN^* and the SSP obtained by adding data points assigned to the maximum score in *ssp_fn* as NSSP (*l_fn* set in line 5 of Listing 1) or to the minimum score in *ssp_fn* (*r_fn* set in line 6 of Listing 1). Both sets are then used in lines 7 and 8 to calculate residuals l^* of r^* by calling the *compute* function on *l_fn* (the former) and *r_fn* (the latter). Out of those two residuals l^* and r^*, the choice which lowers FN^* the most is used to iteratively update SSP and *ssp_fn*. At the end of the process (line 14 of Listing 1), the function builds the SSP (and NSSP, by difference) that contains the most predictions according to a specific ALR.

Listing 2: Pseudocode to compute SSP_{ALR} and $NSSP_{ALR}$ with a divide-and-conquer approach for complex decision functions.

```
1 [SSP_ALR, NSSP_ALR]:
       dcSSP(neg_s, fn_s, ALR, pos, ns)
2   slices = partition(neg_s, fn_s, pos, ns)
3   dacSSP = 0, dacNSSP = 0
4   for each slice in slices do
5     [slice_SSP, slice_NSSP] =
          calcSSP(slice.neg_s, slice.fn_s, ALR,
                  slice.pos)
6     dacSSP = dacSSP + slice_SSP
7     dacNSSP = dacNSSP + slice_NSSP
8   [SSP, NSSP] = calcSSP(neg_s, fn_s, ALR, pos)
9   if dacSSP > SSP then
10    return [dacSSP, dacNSSP]
11  else return [SSP, NSSP]
```

The *compute* function in Listing 1 computes FN^* and SSP according to a given set of NSSP described by the variable *ssp_fn*. First, it computes the FN^* by calling the *residualFN* function (line 16, 24-27 of Listing 1). Then, in lines 17-22 it computes the SSP according to the FN^* quantity:

- should FN^* be zero, this means that the NSSP is empty and therefore all predictions are SSP (lines 17-18);
- if $0 < FN^* \leq$ ALR, some FNs appear in the SSP, but the NSSP is not empty: thus, all data points falling into that area have to be excluded from SSP (lines 19-20);
- lastly, should FN^* be bigger than the ALR, it means that the NSSP embraces all predictions in between the minimum score in *fn_s* and the maximum score in *fn_s* (lines 21-22). In this last case, all FNs and TNs that collocate with, or fall in between FNs are considered NSSP.

This algorithm works well when scores are distributed as in Figures 3a and 3b, Figure 4 or in Figure 5, with a unique continuous range where the binary classifier shows a negative prediction. However, it may sparingly happen that a decision function assigned to a binary classifier derives multiple non-continuous ranges of scores that converted into a

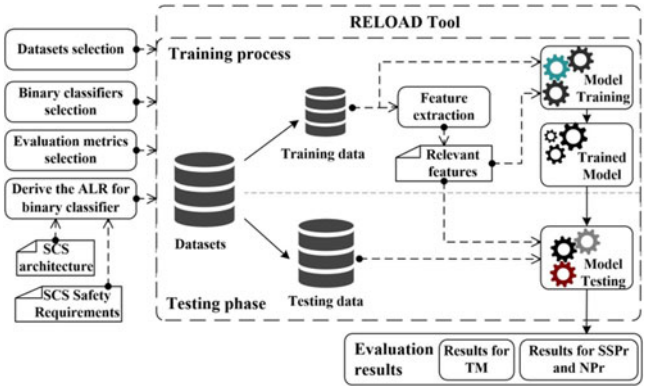

FIGURE 6. Methodology to execute experiments in this work.

negative binary classification. The reader can think about having a decision function for Figure 2 which instead of being "*Negative if* $7 \leq num\_score \leq 9$" is the opposite, or rather "*Negative if* $num\_score < 7$ *or* $num\_score > 9$". In such a context, the algorithm in Listing 1 may end up finding a SSP_{ALR} value which is far from the optimum as the process ends to converge to a single continuous area containing $NSSP_{ALR}$. As such, the algorithm in Listing 1 needs to be adapted to a divide-and-conquer approach we reported in Listing 2. The *dcSSP* algorithm first partitions the output space of algorithm scores into *ns* slices, where each $slice = [neg\_s, fn\_s, pos] \in slices$ contains the negative predictions, the false negatives and the counter for positive predictions for a given portion of the output space. Then, we calculate *calcSSP* from Listing 1 separately for each slice; then, we sum up SSP and NSSP for each slice building the *dacSSP* and *dacNSSP* quantities (rows 5-7 of Listing 2). Those quantities are ultimately (rows 8-11) compared against the SSP and NSSP computed using the *calcSSP* function: the result with the highest SSP is then returned to the user. Noticeably, the *dcSSP* function can be executed recursively: however, to the best of our knowledge, there are no decision functions that may require multiple levels of partitioning of our algorithm to derive adequate SSP and NSSP scores.

B. DEFINING SAFETY-ORIENTED METRICS

Those SSP_{ALR} and $NSSP_{ALR}$ counters we formalized before can then be used to compute metrics based on the distribution of predictions of a binary classifier, as follows:

Sufficiently Safe Prediction rate – SSPr(ALR): is the proportion of predictions that are sufficiently safe to be used by an incorporating SCS to the overall number of predictions with respect to the considered ALR.

$$SSPr(ALR) = \frac{SSP_{ALR}}{NSSP_{ALR} + SSP_{ALR}}$$

No Prediction rate – NPr(ALR): is the proportion of predictions that are not sufficiently safe to be used to the overall number of predictions with respect to the considered acceptable ALR.

$$NPr(ALR) = 1 - \ SSPr(ALR) = \frac{NSSP_{ALR}}{NSSP_{ALR} + SSP_{ALR}}$$

VI. METHODOLOGY AND EXPERIMENTAL SETUP

This section reports on the methodology (shown in Figure 6) and experimental setup we employed in this work.

- We first conducted a comprehensive investigation on public datasets about SCSs to build a solid baseline for our experimental study. Out of existing alternatives, we found a considerable amount of publicly available data about intrusion detection (see Section VI.A).
- Then, we adopted several unsupervised algorithms that have potential in detecting both known and unknown (e.g., zero-day attacks [21]) attacks, which is very important when dealing with intrusions (Section VI.B).
- Besides the two new metrics defined earlier in the paper (with ALR set to 0.01), we have selected the most commonly used evaluation metrics in the literature. The selected metric scores are, further, described in Section VI.C.
- We fed the selected datasets, algorithms, metrics into the RELOAD tool [23], which allows for conducting the experiment (see Section VI.D) according to the experimental setup in Section VI.E.

A. SELECTED DATASETS

Taking advantage of recent surveys such as [24] and querying online portals like *AZSecure – Intelligence and Security Informatics Datasets*, and *UNB – Canadian Institute for CyberSecurity*, we select labelled datasets that contain enough records to guarantee statistical evidence and published in the last decade or very well known. Labels are required for evaluating detection performance of any binary classifier, whereas they are not required for training the unsupervised binary classifiers we use in our experimental study. Our process ended up selecting the following 9 datasets (Table 1), whose references can be found in papers [25], [10].

- NSL-KDD (2009) is a very well-known dataset that contains the following attacks: DoS (Denial-of-Service), R2L (unauthorized access from a remote machine), U2R (unauthorized access to super-user or root functions) and Probing (gather information about a network).
- ISCX12 (2012), generated in a controlled environment based on a realistic network and traffic to depict the real effects of attacks over the network and the corresponding responses of workstations. Four different attack scenarios were simulated by the authors.
- UNSW-NB15 (2015), released by the Australian Defense Force Academy in the University of New South Wales, and it contains 7 attacks that were able to bypass existing security mechanisms.

TABLE 1. Datasets used in this study: Name, release year, data point used, number of ordinal and categorical features, number and percentage of attacks.

Name	Year	# Data Points	Features		Attacks	
			Ord.	Cat.	#	%
ADFANet	2015	132 002	5	6	3	11.3
CICIDS17	2017	200 000	77	5	5	79.7
CICIDS18	2018	200 000	77	5	6	26.2
CIDDS	2015	200 000	5	7	4	14.4
ISCX12	2012	200 000	4	10	4	43.5
NSLKDD	2009	148 516	37	5	4	40.7
SDN20	2020	200 000	63	5	5	66.6
UGR16	2016	200 000	4	6	5	3.3
UNSW-NB15	2015	175 341	38	6	8	6.5

- UGR16 (2016) has been built with real traffic and up-to-date attacks. Its data come from several net-flow collectors strategically located in the network of a Spanish ISP and embeds normal traffic as well as data related to DoS, BotNet, Scan, Blacklist and Spam attacks.
- ADFA-Netflow-IDS (2017), created at the next generation cyber range infrastructure of the Australian Centre of Cyber Security (ACCS) and contains normal and abnormal host (LINUX) and adversarial network activities which authors deemed relevant for future IDS design.
- CIDDS-001 (2017) was created within an emulated small business environment in 2017, contains four weeks of unidirectional flow-based network traffic, and contains several attacks captured from the wild.
- CICIDS17 (2017) was created within an emulated environment over a period of 5 days. For each flow, the authors extracted more than 80 features. The data set contains a wide range of attack types like SSH brute force, Botnet, DoS, DDoS, web and infiltration attacks.
- CICIDS18 (2018). Similar to CICIDS 2017, CICIDS18 was created as an updated version of the previous dataset, containing Brute-Force, Botnet, DoS, DDoS, Web i.e., SQLi, and Infiltration attacks.
- SDN20 (2020) was obtained by monitoring a Software Defined Network installed at the University College Dublin, Ireland. Attacks in SDN include Probe, i.e., network scanning, DDoS, Brute-Force (BFA to bypass the username-password login), and Exploits (privilege escalation known as U2R).

Table 1 provides a summary of the datasets used in this study, including: name, release year, the total size of dataset (# of data points), the size of the subset we selected, the number of features, and frequency of attacks. We used up to 200,000 data points for each dataset: this allows reducing the time needed to execute the whole experimental campaign. Consequently, we used all data points for ADFANet (132 002), NSL-KDD (148 516), UNSW-NB15 (175 341), and cropped others to 200,000 data points. Moreover, we disregarded the usage of categorical features, as it may be not meaningful to compute common operations such as Euclidean distance between two port numbers, or even an average.

B. BINARY CLASSIFIERS

We choose a heterogeneous set of unsupervised binary classifiers to perform intrusion detection on the selected datasets. Different algorithms were chosen out of several families [25], [26], namely clustering, statistical, neighbor-based, angle-based, density-based, classification, and neural networks. We disregard heavy algorithms (e.g., ABOD that has a cubic time complexity), which naturally requires intensive computing and memory resources and therefore may not meet the performance requirements of many systems. Accordingly, we select 12 unsupervised classifiers as follows:

- One algorithm for each family: One-Class SVM (classification family), K-Means (clustering), ODIN (neighbor-based), HBOS (statistical), SOM (neural-network), FastABOD (angle-based), and Sparse Density Observers (SDO, density-based).
- Other well-known binary classifiers as COF, LOF, LDCOF, Isolation Forests (iForest), and G-Means to widen the selection of unsupervised algorithms.

C. EVALUATION METRICS

Besides the new metrics we calculated by using an ALR of 0.01, thus SSPr(0.01) and NPr(0.01), we considered the most commonly used performance metrics such as Accuracy (ACC), Precision (P), Recall (R), False Positive Rate (FPR), F1-Score (F1 or F-Measure). Moreover, Matthews Coefficient (MCC) is a correlation coefficient between the true and predicted classes, and achieves a high value only if the classifier obtains good results in all the entries of the confusion matrix, and can be calculated relying on the following equation

$$\mathrm{MCC} = \frac{\mathrm{TP} * \mathrm{TN} - \mathrm{FP} * \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$$

The MCC is bounded to [-1; 1], where a value of 1 represents a perfect prediction, 0 random guessing, and -1 total disagreement between prediction and observation. The Area Under Receiver Operating Characteristics (ROC) Curve (AUC) and the Area Under the Convex Hull of the ROC Curve (AUCH) represent the degree of the separability of the ROC curve, and evaluate the tradeoff between TPR and FPR of a classifier.

We also considered several under-used metrics like Gini index that is used to measure how well a model is classifying points, and normalize the AUC so that a random classifier scores 0, and a perfect classifier scores 1, as follows: Gini = 2 x AUC -1. H-measure (H) has been developed to overcome the situation of incurring different misclassification costs for different classifiers, and it measures of the expected minimum loss obtained for a given cost distribution. H-measure takes a severity ratio as input, which examines how much more severe misclassifying one class instance is than misclassifying another class instance. Kappa Statistics (KS)

TABLE 2. A portion of the results (metric scores) of applying the binary classifiers to the datasets, ordered by decreasing SSPr. Highlighted cases are those that are being explored through plots in this section.

Case ID	Binary Classifier	Dataset	Traditional Metrics															Distribution-Based		
			FN %	FPR	P	R	F1	F2	MCC	ACC	AUC	AUCH	H	Gini	KS	Youden	PR Gain	SSPr (0.01)	MCC (0.01)	ACC (0.01)
1	FastABOD	ADFANet	0.01	0.010	0.977	0.999	0.988	0.995	0.983	0.993	0.99	1.00	0.98	0.99	0.99	0.99	0.98	100.00	0.98	0.99
2	LOF	ADFANet	0.00	0.079	0.851	1.000	0.919	0.966	0.885	0.946	0.97	0.97	0.85	0.93	0.94	0.00	0.85	100.00	0.89	0.95
3	SVM	ADFANet	0.00	1.000	0.310	1.000	0.473	0.692	0.002	0.310	0.59	0.79	0.51	0.19	0.58	0.00	0.31	100.00	0.00	0.31
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
15	LOF	SDN20	0.01	0.900	0.691	0.999	0.817	0.918	0.262	0.701	0.56	0.71	0.36	0.12	0.43	0.00	0.69	100.00	0.26	0.70
16	iForest	SDN20	0.00	0.232	0.896	1.000	0.945	0.977	0.829	0.923	0.99	0.99	0.93	0.99	0.95	0.54	0.90	100.00	0.83	0.92
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
19	SOM	UNSW	0.01	1.000	0.379	0.999	0.550	0.753	0.000	0.379	0.56	0.65	0.09	0.13	0.20	0.02	0.38	100.00	0.00	0.38
20	SOM	SDN20	0.13	0.878	0.696	0.998	0.820	0.918	0.281	0.707	0.92	0.95	0.75	0.84	0.79	0.01	0.70	99.87	0.29	0.71
21	SVM	SDN20	0.13	0.894	0.693	0.998	0.817	0.917	0.261	0.702	0.92	0.95	0.76	0.84	0.80	0.00	0.69	99.82	0.26	0.70
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
31	ODIN	SDN20	0.12	0.395	0.835	0.990	0.906	0.955	0.691	0.862	0.60	0.74	0.39	0.21	0.45	-0.11	0.83	96.10	0.65	0.86
32	SDO	CIDDS	0.20	0.611	0.510	0.996	0.674	0.836	0.440	0.625	0.56	0.71	0.19	0.12	0.37	-0.34	0.51	95.92	0.40	0.60
33	SOM	CIDDS	0.08	0.783	0.448	0.998	0.619	0.801	0.308	0.521	0.58	0.72	0.20	0.16	0.43	0.00	0.45	95.55	0.21	0.48
34	SVM	CIDDS	0.08	0.781	0.449	0.998	0.619	0.801	0.308	0.522	0.58	0.72	0.20	0.16	0.43	0.00	0.45	95.53	0.21	0.48
35	FastABOD	SDN20	0.09	0.290	0.873	0.995	0.930	0.968	0.778	0.900	0.82	0.88	0.49	0.64	0.64	0.31	0.87	95.18	0.73	0.90
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
41	KMEANS	UNSW	0.73	0.765	0.439	0.980	0.607	0.787	0.290	0.518	0.63	0.68	0.13	0.26	0.24	0.00	0.44	91.74	0.01	0.44
42	GMEANS	UNSW	1.24	0.746	0.443	0.969	0.608	0.783	0.289	0.526	0.52	0.62	0.11	0.04	0.22	0.00	0.44	91.34	0.10	0.45
43	LOF	UNSW	6.58	0.811	0.384	0.827	0.525	0.672	0.220	0.651	0.57	0.64	0.08	0.14	0.18	0.16	0.43	90.32	0.22	0.63
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
66	iForest	ADFANet	0.03	0.397	0.713	0.984	0.827	0.915	0.636	0.794	0.84	0.89	0.65	0.69	0.77	0.58	0.71	69.08	0.00	0.71
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
77	ODIN	CICIDS18	1.97	0.075	0.972	0.950	0.946	0.951	0.876	0.938	0.67	0.97	0.81	0.94	0.87	0.74	0.90	51.30	0.00	0.93
78	ODIN	ISCX	0.80	0.468	0.139	0.872	0.240	0.425	0.219	0.559	0.69	0.77	0.04	0.37	0.45	0.00	0.13	50.03	0.00	0.14
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

measures the goodness of fit for the continuous cumulative distribution of data samples. Youden Index measures the effectiveness of a diagnostic classifier as well as selecting an optimal threshold value for it, and it can be calculated, as follows: Youden Index = sensitivity + specificity -1. Finally, the Precision-Recall-Gain curve (PR-Gain) plots Precision Gain on the y-axis against Recall Gain on the x-axis in the unit square (i.e., negative gains are ignored), and it is calculated based on enclosing an area that is directly related to the expected F1-score in a similar way as AUC is related to expected accuracy. Additionally, we computed ACC(0.01) and MCC(0.01), the Accuracy and Matthews Coefficient restricted to the $SSP_{0.01}$ predictions areas. Those two metrics provide a quantitative measure about the detection performance of binary classifiers when they are providing sufficiently safe predictions, without accounting for $NSSP_{0.01}$.

D. TOOLING

RELOAD [23] is an open-source tool that offers a simple GUI, and includes the implementations for all unsupervised algorithms described in Section VI.B. Concerning the selected datasets (Section VI.A), we have downloaded their source files from repositories and re-shaped them as CSV files, which can be processed by RELOAD. RELOAD also includes all metrics implementations described in Section VI.C but H-Measure and AUCH and the two new ones (NPr and SSPr). Therefore, we relied on an R-package[5] to calculate H-Measure and AUCH, and we have extended RELOAD with the implementations of the new metrics as well as a technique to set the required thresholds required for calculating them based on the specified ALR that has to be entered as an input by the user.

Additionally, RELOAD embeds automatic tuning of parameters of algorithms through grid searches and facilitates examining outputs through graphical plots. Moreover, we used Information Gain [27] as a feature selection strategy and F-Score(2) as a target metric for parameter tuning which is automatically calculated by the tool through grid searches.

E. EXPERIMENTAL SETUP

After setting up RELOAD, preparing the selected datasets, algorithms, and metrics, we run the experiments with a 50-50 train-test split, 10-fold cross-validation and collected metric scores. Noticeably, RELOAD automatically removes labels (if any) from the training set, as unsupervised ML algorithms can perform training while being completely unaware of labels themselves. The experiments have been executed on a server equipped with Intel Core i7-6700 with four 3.40 GHz cores, 24GB of RAM and 100GB of user

5.Measuring classification performance: the H-measure package. https://cutt.ly/dccy3P2

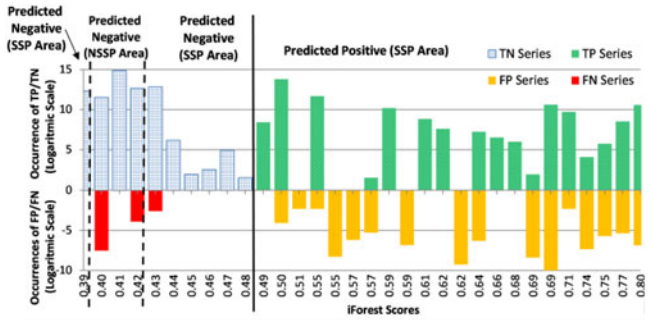

FIGURE 7: Bar chart showing the distribution of predictions of the iForest algorithm to the ADFANet dataset (ID 66).

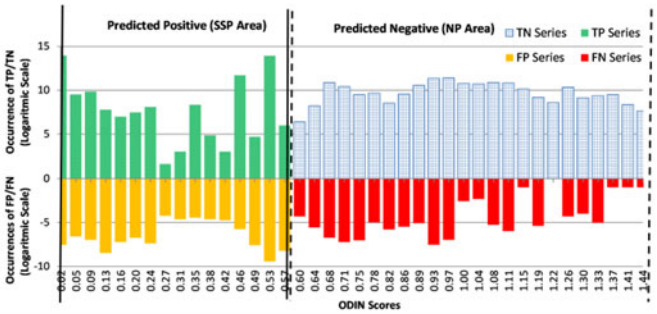

FIGURE 8: Bar chart showing the distribution of predictions of ODIN to CICIDS18 - ID 77 - (up) and ISCX – ID 78 - (down) datasets.

storage. Executing all the experiments required more than three weeks of 24H execution. Applying the 12 binary classifiers to the 9 datasets resulted in 108 cases. The results of the experiment performed including all metric scores, RELOAD data logs and snapshots showing the distribution of results of the application of the algorithms to the datasets (e.g., TP, TN, FP and FN) can be found at https://bit.ly/3aysI4i.

VII. EXPERIMENTAL RESULTS AND DISCUSSION

This section is dedicated to the discussion of the results, and it is organized as follows: Section VII.A discusses the properness of incorporating binary classifiers into SCS concerning the specified ALR, followed in Section VII.B by a discussion on whether we can assess such properness using traditional metrics. Section VII.C debates on the correlation of traditional metrics with respect to metrics based on distribution.

A. ANALYSIS OF DISTRIBUTION-BASED METRICS

Table 2 shows a portion of the results of applying binary classifiers to the datasets, ordered by decreasing SSPr(0.01).

Applying the iForest algorithm to the ADFANet dataset (ID 66 in Table 2) resulted in an SSPr(0.01) of 69.08%: this means that roughly 69% of the predictions of this case are sufficiently safe to be used by the incorporating SCS. Accordingly, the NPr is 30.92%, i.e., 30.92% of the predictions of this case are not sufficiently safe to be used in a SCS. Figure 7 shows the logarithmic scale distribution of TP, FP, FN, and TN concerning this case; solid vertical bars represent boundaries of the decision function, while dashed bars show boundaries of the NSSP area. From left to right, the figure shows four different areas:

i. Predicted Negative area that contains data points that have been predicted negative but nevertheless can be safely used in a SCS (SSP).
ii. Predicted Negative that contains all data points that have been predicted negative and cannot be safely used (NSSP) in a SCS; and
iii. Predicted Negative area that contains data points that have been predicted negative but nevertheless can be safely used in a SCS (SSP).
iv. Predicted Positive that contains data points that have been predicted positive by the binary classifier and can be safely used by the incorporating SCS (SSP).

This case is ranked 66/108 accounting only for the safety metric SSPr with an ALR of 0.01. In fact, we see that the area ii) between dashed lines in Figure 7 comprises many data points that are thus considered as NSSP and that cannot be safely used in a SCS as they co-locate with FNs (see red bars in the figure). Noticeably, this binary classifier generates very few FNs, ending up with a very high Recall of 99.2. However, many FNs co-locate with TNs, which are considered NSSP as well.

There are many other cases that end up having very poor SSPr even with relatively low FN% and high Recall. It is the case of the application of the ODIN algorithm to the CICIDS18 dataset (ID 77 in Table 2), which produced 1971 FNs out of 100000 data points (1.97% of all predictions). However, due to the scattered distribution of such FNs, almost half of the overall predictions are considered not sufficiently safe to be by the incorporating SCS, i.e., NPr is 48.7% and SSPr is 51.3%. The distribution of this case is shown on top of Figure 8, where we clearly see how all the predicted negative predictions end up being NSSP. Another example of the effect of the scattered distribution of FNs is the application of the ODIN algorithm to the ISCX dataset (ID 78 in Table 2), where the FNs are only 0.8% of the data points in the test set, yet the SSPr was 50.0%. Again, down in Figure 8 we see how also in this case the distribution of FNs (red bars) overlaps completely with TNs (blue patterned bars), which all become NSSP.

Yet another interesting case (ID 43 in Table 2) is provided by the application of the LOF binary classifier to the UNSW dataset, whose distribution is shown in Figure 9. LOF generates many FNs in this dataset with a FN% of 6.58; however, those FNs are mainly distributed in a relatively small area in between the dashed vertical lines in the figure. As a result, all data points for which LOF assign scores in the range 0.85 and 2.85 are SSP as they are predicted positive, whereas data points for which LOF assign scores bigger than 5 are SSP as well even if the binary classifier will predict negative label. FNs appear in this last area, but they are overall below the

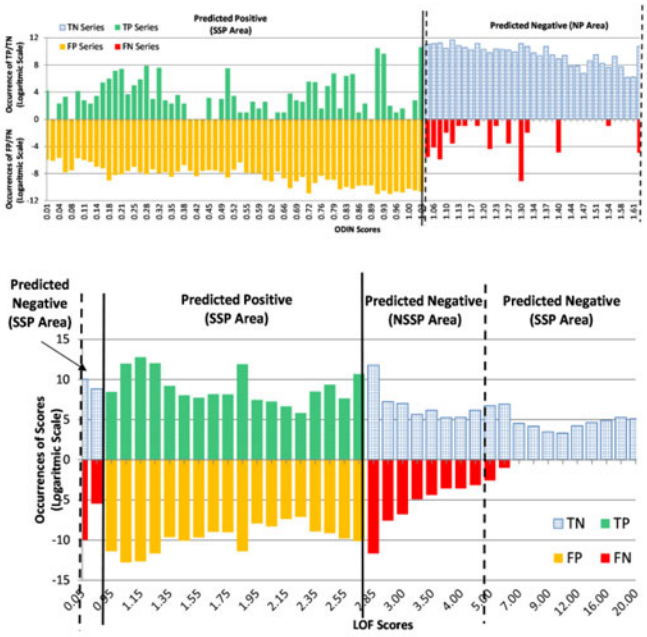

FIGURE 9: Bar chart showing the distribution of predictions when applying LOF algorithm to the UNSW dataset (ID 43).

ALR = 0.01 that we used in our experiments and thus are considered tolerable.

In turn, there are other cases (see IDs from 1 down to 19 in Table 2) where all predictions are sufficiently safe, thus SSPr is 100%. In those cases, the FN* (or FN% in Table 2) is lower than 0.01, which is the ALR we set for this experimental study. As such, the residual FNs are considered acceptable without major safety implications.

Noticeably, the last two columns of Table 2 report on MCC and Accuracy values that are calculated only for the SSP area. The safety specialist is for sure interested in understanding the performance of a binary classifier regarding safety, but it is also interested in understanding how well the classifier performs in the so-called safe area where all SSP lie. When SSPr(0.01) = 100%, those scores equal to the regular MCC and Accuracy scores. However, it is common to have a quite evident decrease in those metric values: when FNs co-locate with many TNs, both predictions fall into the NSSP and thus lower both the overall amount of FN (which is desirable) and TN, which instead negatively affects the overall classification performance of a binary classifier.

B. ANALYSIS OF TRADITIONAL METRICS

This section elaborates on the contribution of traditional metrics in understanding if a binary classifier can be safely incorporated into a SCS. Table 2 already reports on traditional metric scores, but cannot provide an overall comparison of them. Therefore, we first observe that there are 43 cases – the last one being the LOF algorithm on the UNSW dataset at ID 43 in Table 2 - in which the SSPr(0.01) is above 90%. This means that for those 43 cases we have no more than 10% of the overall predictions in the NSSP according to the distribution analysis. Then, we scan traditional metric scores to derive the "best" 43 cases for each of those metrics. This allows drawing Table 3, which contains the number of cases that have both a SSPr(0.01) bigger than 90% and that appear in the best 43 values for a given traditional metric. In a nutshell, the higher the overlap, the more similar a metric is with respect to SSPr.

Among the top 43 best scoring cases concerning Recall, 36 of them have SSPr(0.01) $\geq$ 90%; 18 cases appeared in the top 43 best scoring cases when we applied F-Score(2), while other metrics show even less overlap. It is easy to note that more than half of the cases where SSPr(0.01) $\geq$ 90% did not appear within the top scoring cases for all traditional metrics but Recall. The similarity between SSPr and Recall can be explained as follows: Recall focuses on the proportion of TPs overall positives (both TPs and FNs), yet it does not account for their distribution. We already highlighted in Figures 7 and 9 how a scattered distribution of FNs may translate to a very high Recall but low SSPr(0.01) due to many FNs that co-locate with TNs (Figure 7), or a binary classifier with quite poor Recall which instead accumulates FNs in restricted ranges and thus have high SSPr scores.

C. CORRELATION BETWEEN TRADITIONAL METRICS AND DISTRIBUTION-BASED METRICS

To provide an even better understanding of the correlation between traditional metrics and metrics based on distribution, Table 4 reports the R-Squared (R2) correlations of the metrics across all 108 cases, colored with a gradient that reflects the strength of such correlation: the darker, the stronger.

We can notice how groups of metrics as {AUC, AUCH, Gini, H-Measure}, {Precision, F1, F2, PR-Gain}, and {FPR, MCC, KS, Youden, ACC} emerge as tightly correlated among themselves. More importantly, the table allows quantitatively evaluating whether and to which extent SSPr is correlated with traditional metrics. It is easy to note that SSPr is only moderately correlated with Recall.

Importantly, we do not suggest that metrics based on distribution should replace traditional ones when dealing with SCSs but rather used in conjunction. It turns out evident from this analysis that SSPr catches aspects of the behavior of a binary classifier that are different from existing ones and therefore should be considered when investigating the

TABLE 3. Number of cases that result in a SSPr(0.01) $\geq$ 90 and that correspond to the best scoring cases for traditional metrics.

R	F2	Youden	H	F1	PR-Gain	AUC	AUCH	Gini	P	KS	MCC	ACC	FPR
36/43	18/43	16/43	16/43	14/43	13/43	10/43	10/43	10/43	10/43	9/43	6/43	6/43	4/43

IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING

TABLE 4. R-Squared correlations of different metrics. Dark background points to cells with strong correlations.

	FPR	P	R	F1	MCC	ACC	AUC	AUCH	H	Gini	KS	Youden	PRGain
P	0.32												
R	0.46	0.00											
F1	0.17	0.89	0.03										
MCC	0.63	0.76	0.06	0.68									
ACC	0.85	0.60	0.23	0.43	0.78								
AUC	0.26	0.38	0.01	0.39	0.41	0.38							
AUCH	0.28	0.48	0.00	0.49	0.50	0.45	0.92						
H	0.10	0.58	0.04	0.66	0.44	0.27	0.67	0.76					
Gini	0.26	0.38	0.01	0.39	0.41	0.38	1.00	0.92	0.67				
KS	0.71	0.71	0.10	0.62	0.98	0.84	0.42	0.50	0.41	0.42			
Youden	0.71	0.53	0.12	0.49	0.91	0.76	0.39	0.45	0.30	0.39	0.94		
PR-Gain	0.10	0.84	0.08	0.97	0.59	0.32	0.33	0.44	0.65	0.33	0.52	0.40	
SSPr(0.01)	0.41	0.03	0.61	0.00	0.14	0.27	0.06	0.05	0.00	0.06	0.18	0.21	0.00

properness of incorporating a binary classifier into a SCS. On the other side, there are cases that show a perfect (100%) SSPr but generate many misclassifications, mainly FPs. For example, compare the applications of the FastABOD and SVM algorithms to the ADFANet dataset (IDs 1 and 3 in Table 2 respectively). They both achieve SSPr of 100, but the former shows an accuracy of 0.993 while the latter achieves an accuracy of only 0.310 (and MCC = 0). This last case highlights how the training phase of the SVM algorithm on ADFANet dataset devised a model which always predicts positive: therefore, there are no FNs (and thus Recall = SSPr(0.01) = 100%) but also there are no TNs, with all the predictions ending up being either FPs or TPs.

VIII. RELATED WORKS

Several studies have been proposed suggesting methodologies, approaches, or processes that focus on the safe incorporation of binary classifiers into SCS. For example, Varshney and Alemzadeh [28] discussed four main categories of approaches/strategies for increasing safety in ML namely: inherently safe design, safety margins, safe fail, and procedural safeguards. Cheng *et al.* [29] proposed the nn-dependability-kit that is an open-source tool, which can be used to support data-driven engineering of Neural Networks (NN) for safety-critical systems.

Borg *et al.* [11] conducted a review concerning the state-of-the-art verification and validation methods for automotive systems that rely on ML in general and Deep learning Neural Networks (DNN) in particular. The authors identified various key challenges concerning the use of safety-critical DNN components in the automotive domain. Moreover, Henriksson *et al.* [30] conducted a preliminary exploratory study that tried to identify which parts of the ISO 26262 - Road vehicles – Functional safety standard need to be adapted to allow safety-critical ML development in the automotive context.

Shafaei *et al.* [31] investigated several key challenges in ensuring safety for ML methods in the autonomous driving domain. They mainly focused on the uncertainty of prediction in ML performance and considered when it might be originated by i) the ML algorithm (model-dependent); and/or ii) the training data (data-dependent). They categorized safety-critical situations originated from this issue into four different cases, and they propose suggested techniques to address the challenges in each case.

Burton *et al.* [32] discussed recent challenges involved in assuring the safety of Highly Automated Driving (HAD) functionalities that rely on binary classifiers, and they proposed applying an assurance case approach to argue the safety of such HADs. Moreover, Sherin *et al.* [33] conducted a systematic mapping concerning the testing of ML algorithms aiming to provide a detailed discussion on research gaps and future recommendations. Based on their study, they concluded that although there is a significant increase in the number of publications concerning the testing of ML, only a few tools are publicly available and there is not enough empirical evidence to assess and compare them.

Picardi *et al.* [34] developed several "general" patterns to be used while developing assurance arguments for demonstrating the safety of the ML components deployed in a SCS, and they proposed a process to be followed in each stage of the ML lifecycle to create the assurance cases for ML components. Gauerhof *et al.* [35] discussed the assurance of the safe performance of Machine Learnt Models (MLMs) that are operating in the automotive domain. They proposed a five stages lifecycle process concerning MLM that includes requirements elicitation, data management, model learning, model verification and model deployment. Further, they evaluate their proposal against a pedestrian detection at crossings example.

On the other hand, Biondi *et al.* [36] propose a visionary software architecture that allows using deep learning algorithms in safety-critical systems by integrating diverse technologies (e.g., hypervisors, run time monitoring, redundancy with diversity, fault recovery). However, the authors listed and discussed several challenges that hinder the realization of the proposed architecture. Hossin and Sulaiman [37] systematically reviewed and highlight the shortcomings of various metrics (e.g., accuracy, sensitivity, recall) that are used for evaluating classifiers. Moreover, the authors suggested several aspects that must be considered when constructing a new metric for classification such as the metric should not be too complex to be implemented, has a feasible computational cost, can favor the minority class, etc. Hicks *et al.* [38] conducted a study on how various metrics can be interpreted differently depending on the context and the purpose of the study. To demonstrate that, they selected five different studies and they recalculate the metrics used in these studies, and also calculate other metrics (called "missing" that includes less commonly used metrics such as Negative Predictive Value (NPV), Threat Score (TS) that can be derived from the

confusion matrix) in each of selected studies. The authors concluded that using only a subset of metrics could give a false impression of a model's actual performance, and explained how the studied metrics should be interpreted.

Sheh, R. [39] criticizes the use of traditional ML metrics for evaluating the performance of ML systems incorporated in intelligent robots., and presented several aspects of performance that should be considered when evaluating ML systems to avoid being misled by the results of traditional metrics, including: 1- failure details that aims at understanding where failures happen, which is more important than just measuring the failure rate; 2- trade-off invariant performance measures as there are inevitably tradeoffs to be made between different metrics of performance (e.g., FPs and FNs); 3- important and unimportant failures, as some failures in performance may be easily corrected by decision-making stages and, thus, optimizing for them represents a wasted trade-off (e.g., FP in our study); and 4- other characteristics (e.g., explainability, transparency, interpretability, and correctability). In the same line, Salman *et al.* [40] suggest that any metric for evaluating an ML model built for a security application should satisfy two requirements: 1- the mistakes made by misclassifying attack and benign activities should not be treated equally; and 2- correctly classifying a sample in a class should not be the same as a mistake made by misclassifying in the same class. The authors further developed a metric, namely Safety Score that satisfies the two requirements, and extends the accuracy metric by adding weights to each of the four classes of the confusion metrics.

Aydemir, O. [41] proposes a new metric for evaluating the performance of a classifier, namely polygon area metric (PAM) that uses six traditional metrics including accuracy, sensitivity, specificity, AUC, Jaccard index, and F-measure to generate a polygon, then, calculates its area that produces the result of PAM. However, PAM is a general-purpose evaluation metric that was not developed with safety in mind. Moreover, its six considered metrics are basically based on the four classes of the confusion matrix. Finally, Samake and Boulmane [42] propose to evaluate the reliability of deep learning models through the concept of acceptance, rejection, and abstain zones that can be defined through two thresholds. Then, the prediction is accepted, for the acceptance zone. Otherwise, it is rejected for the reject zone and not considered for the abstain zone. This approach, although close to the idea of our earlier work [43], does not consider how the predictions are distributed. Moreover, when applied by the authors, its results were not promising as there were very few predictions in the acceptance as well as the rejection zones. Accordingly, almost all predictions fall within the abstain zone.

Most of these works provide general solutions (one size fits all) without considering the special aspects/characteristics of the domain of disclosure, which may heavily influence the desired safety requirements to be achieved by the ML-based component. As discussed earlier, some incorrect predictions can be of higher cost for some SCS since they may lead to catastrophic incidents. Therefore, any solution for the safe incorporation of ML algorithms into SCS should consider the special aspects/characteristics of the domain of disclosure.

IX. CONCLUDING REMARKS

A. THREATS TO VALIDITY

We discuss here the different threats to the validity of this study:

- *Internal validity* is concerned with factors that may have influenced the investigated factors, but they have not been considered in the study. In particular, it would be interesting to elaborate on how the completeness of the features and their balanced distribution in the training set may influence the performance of binary classifiers.
- *External validity* is concerned with to what extent the results of the study can be generalized. Our study started by motivating an analysis of binary classifiers based on the distribution of their scores, which applies to any binary classifier. In the experimental evaluation, we considered 12 unsupervised binary classifiers that have been applied to 9 public attack datasets and did not account for supervised nor semi-supervised ML algorithms nor datasets from other domains. However, the choice of the binary classifiers and datasets related to SCSs for experiments was needed only to prove the importance of distribution-based analysis. Therefore, our choices in the experimental campaign do not limit the generalization of our findings.
- *Reliability validity* is concerned with to what extent the study is dependent on the researcher(s), i.e., if other researchers conducted the exact same study, the result should be almost the same. The results of the experiment including all metric scores are publicly available at https://bit.ly/3aysI4i, and any researcher can repeat the experiments and she/he should get similar results.

B. CONCLUSION

In this paper, we discussed and experimentally tried to assess the properness of incorporating ML algorithms that perform binary classification into Safety-Critical Systems using various commonly used metrics as well as two new metrics we defined, namely SSPr and NPr. We carefully analyzed the overall results of the experiment to widen our knowledge about how we can assess the properness of incorporating binary classifiers in SCS. We have empirically demonstrated the applicability of the SSPr and NPr metrics, and based on their scores, the performance of binary classifiers in 43 out of the overall 108 cases showed the potential to be incorporated into the SCS used as our case study. Moreover, it was clear that commonly used metrics may not be appropriate for assessing the performance of binary classifiers concerning safety. Additionally, such performance might be highly influenced by the dataset they are applied to. Our study focused mainly on FNs as the type of incorrect prediction that has the highest cost and should be avoided. However, we cannot

exclude the idea that for some SCS both types of incorrect predictions (FN and FP) might have the same cost, or even only FP might be the incorrect prediction of the highest cost for other SCS. To evaluate the performance of binary classifiers for such systems, the developed metrics can be tuned concerning the incorrect prediction(s) of interest.

C. FUTURE WORKS

For future work, we aim to deeply investigate the main reasons for the scattered distribution of FNs. Understanding this problem will allow minimizing the NSSP area, which may significantly improve the performance of binary classifiers concerning safety. We intend also to understand how we can increase the contrast between the scores of negative and positive predictions [44], which might improve the performance since it separates better those two classes. As binary classifiers are trained on data that are susceptible to data poisoning attacks, whereby malicious users inject false data points intending to skew the learned model, we intend to investigate how we can assure that the used dataset(s) are not compromised in a way that influences our proposed approach. Building on our work concerning meta-learning [45], we are planning to extend our proposed approach beyond the evaluation of simple classifiers to evaluate the performance of N-version binary classifiers concerning safety. Finally, we want to research how feature selection impacts the performance of binary classifiers concerning safety.

REFERENCES

[1] J. Schumann, P. Gupta, and Y. Liu, "Applications of neural networks in high assurance systems," in *Neural Networks*, Berlin, Germany: Springer, 2010, pp. 1–19.

[2] M. Bozzano and A. Villafiorita, *Design and Safety Assessment of Critical Systems*, Boca Raton, FL, USA: Auerbach Pub, 2010.

[3] M. Gharib, P. Lollini, M. Botta, E. Amparore, S. Donatelli, and A. Bondavalli, "On the safety of automotive systems incorporating machine learning-based components: A position paper," in *Proc. 48th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops*, 2018, pp. 271–274.

[4] A. Taylor, S. Leblanc, and N. Japkowicz, "Anomaly detection in automobile control network data with long short-term memory networks," in *Proc. 3rd IEEE Int. Conf. Data Sci. Adv. Analytics*, 2016, pp. 130–139.

[5] D. M. Powers, "Evaluation: From precision, recall and f-measure to ROC, informedness, markedness and correlation," *J. Mach. Learn. Technol.*, vol. 2, no. 1, pp. 37–63, 2011.

[6] M. Gharib, T. Zoppi, and A. Bondavalli, "Understanding the properness of incorporating machine learning algorithms in safety-critical systems," in *Proc. 36th Annu. ACM Symp. Appl. Comput.*, 2021, pp. 232–234.

[7] C. Miller and C. Valasek, *Remote Exploitation of an Unaltered Passenger Vehicle*. Isanti, MN, USA: Black Hat, 2015, pp. 1–91. [Online]. Available: http://Illmatics.Com/Remotecarhacking.Pdf

[8] M. Hanselmann, T. Strauss, K. Dormann, and H. Ulmer, "CANet: An unsupervised intrusion detection system for high dimensional can bus data," *IEEE Access*, vol. 8, pp. 58194–58205, 2020.

[9] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan.–Mar. 2004.

[10] T. Zoppi, A. Ceccarelli, and A. Bondavalli, "Unsupervised algorithms to detect zero-day attacks: Strategy and application," *IEEE Access*, vol. 9, pp. 90603–90615, 2021.

[11] M. Borg *et al.*, "Safely entering the deep: A review of verification and validation for machine learning and a challenge elicitation in the automotive industry," *J. Automot. Softw. Eng.*, vol. 1, pp. 1–19, 2019.

[12] R. Salay and K. Czarnecki, "Using machine learning safely in automotive software: An assessment and adaption of software process requirements in ISO 26262," Aug. 2018.

[13] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE Int. J. Transp. Saf.*, vol. 4, no. 1, pp. 15–24, 2016.

[14] C. Dukes, "Committee on national security systems (CNSS) glossary," CNSSI, Fort Meade, MD, Tech. Rep. 4009, 2015.

[15] International Electrotechnical Commission, "Functional safety of electrical/electronic/programmable electronic safety-related systems," IEC, Geneva, Switzerland, Tech. Rep. IEC 61508, 2000.

[16] CENELEC, "EN 50129:2003 CLC/TR 50404 railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling," British Standards Institution, London, U.K., Tech. Rep. CLC/TR 50404, 2003.

[17] ISO, ISO, "26262-6: 2018—road vehicles—functional safety—part 6: Product development at the software level," 2018. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:26262:-6:ed-2:v1:en

[18] T. Kelly and R. Weaver, "The goal structuring notation-A safety argument notation," in *Proc. Dependable Syst. Netw. Workshop Assurance Cases*, 2004, pp. 1–6.

[19] C. M. Bishop, *Pattern Recognition and Machine Learning*, Berlin, Germany: Springer, 2006.

[20] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," *ACM SIGMOD Rec.*, vol. 29, pp. 427–438, 2000.

[21] S. Checkoway *et al.*, "Comprehensive experimental analyses of automotive attack surfaces," in *Proc. 20th USENIX Secur. Symp.*, 2011, pp. 77–92.

[22] X. Wan, W. Wang, J. Liu, and T. Tong, "Estimating the sample mean and standard deviation from the sample size, median, range and/or interquartile range," *BMC Med. Res. Methodol.*, vol. 14, no. 1, pp. 1–13, 2014.

[23] T. Zoppi, A. Ceccarelli, and A. Bondavalli, "Evaluation of anomaly detection algorithms made easy with reload," in *Proc. Int. Symp. Softw. Rel. Eng.*, 2019, pp. 446–455.

[24] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Comput. Secur.*, vol. 86, pp. 147–167, 2019.

[25] T. Zoppi, A. Ceccarelli, T. Capecchi, and A. Bondavalli, "Unsupervised anomaly detectors to detect intrusions in the current threat landscape," *ACM/IMS Trans. Data Sci.*, vol. 2, no. 2, pp. 1–26, 2021.

[26] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PLoS One*, vol. 11, no. 4, 2016, Art. no. E0152173.

[27] B. Azhagusundari and A. S. Thanamani, "Feature selection based on information gain," *Int. J. Innov. Technol. Exploring Eng.*, vol. 2, no. 2, pp. 18–21, 2013.

[28] K. R. Varshney and H. Alemzadeh, "On the safety of machine learning: Cyber-physical systems, decision sciences, and data products," *Big Data*, vol. 5, no. 3, 2017, pp. 246–255.

[29] C. H. Cheng, C. H. Huang, and G. Nuhrenberg, "NN-dependability-kit: Engineering neural networks for safety-critical autonomous driving systems," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2019, pp. 1–6.

[30] J. Henriksson, M. Borg, and C. Englund, "Automotive safety and machine learning: Initial results from a study on how to adapt the ISO 26262 safety standard," in *Proc. Int. Conf. Softw. Eng.*, 2018, pp. 47–49.

[31] S. Shafaei, S. Kugele, M. H. Osman, and A. Knoll, "Uncertainty in machine learning: A safety perspective on autonomous driving," in *Lecture Notes In Computer Science*. Berlin, Germany: Springer, 2018, pp. 458–464.

[32] S. Burton, L. Gauerhof, and C. Heinzemann, *Making the Case for Safety of Machine Learning in Highly Automated Driving*. Berlin, Germany: Springer, 2017, pp. 5–16.

[33] S. Sherin, M. Uzair Khan, and M. Zohaib Iqbal, "A systematic mapping study on testing of machine learning programs," 2019, *arXiv:1907.09427*.

[34] C. Picardi and C. Paterson, R. D. Hawkins, and R. Calinescu, and I. Habli, "Assurance argument patterns and processes for machine learning in safety-related systems," in *Proc. Workshop Artif. Intell. Saf.*, 2020, pp. 23–30.

[35] L. Gauerhof, R. Hawkins, C. Picardi, C. Paterson, Y. Hagiwara, and I. Habli, "Assuring the safety of machine learning for pedestrian detection at crossings," in *Proc. 39th Int. Conf. Comput. Saf. Rel. Secur.*, 2020, pp. 197–212.

[36] A. Biondi, F. Nesti, G. Cicero, D. Casini, and G Buttazzo, "A safe, secure, and predictable software architecture for deep learning in safety-critical systems," *IEEE Embed. Syst. Lett.*, vol. 12, no. 3, pp. 78–82, Sep. 2020.

[37] M. Hossin and M. N. Sulaiman, "A review on evaluation metrics for data classification evaluations," *Int. J. Data Mining Knowl. Manage. Process*, vol. 5, no. 2, pp. 1–11, 2015.
[38] S. A Hicks *et al.*, "On evaluation metrics for medical applications of artificial intelligence," *Sci. Rep.*, vol. 12, no. 1, pp. 1–9, Apr. 2022.
[39] R Sheh, "Evaluating machine learning performance for safe, intelligent robots," in *Proc. IEEE Int. Conf. Intell. Saf. Robot.*, 2021, pp. 388–393.
[40] T. Salman, A. Ghubaish, D. Unal, and R Jain, "Safety score as an evaluation metric for machine learning models of security applications," *IEEE Netw. Lett.*, vol. 2, no. 4, pp. 207–211, Dec. 2020.
[41] O Aydemir, "A new performance evaluation metric for classifiers: Polygon area metric," *J. Classification*, vol. 38, no. 1, pp. 16–26, Apr. 2021.
[42] A. Samake and L Boulmane, "Acceptance and rejection zones for a classifier's predictions in deep learning," in *Proc. 5th Int. Conf. Intell. Comput. Data Sci.*, 2021, pp. 1–5.
[43] M. Gharib and A Bondavalli, "On the evaluation measures for machine learning algorithms for safety-critical systems," in *Proc.15th Eur. Dependable Comput. Conf.*, 2019, pp. 141–144.
[44] H. P. Kriegel, P. Kroger, E. Schubert, and A Zimek, "Interpreting and unifying outlier scores," in *Proc. 11th Siam Int. Conf. Data Mining*, 2011, pp. 13–24.
[45] T. Zoppi, M. Gharib, M. Atif, and A Bondavalli, "Meta-learning to improve unsupervised intrusion detection in cyber-physical systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 5, no. 4, Oct. 2021, Art. no. 42.

MOHAMAD GHARIB received the PhD degree from the Department of Information Engineering and Computer Science, University of Trento, Italy, in 2015. He is a lecturer of software engineering with the Institute of Computer Science, University of Tartu. He was a postdoctoral researcher with the University of Florence. Before that, he was a postdoctoral researcher with the Department of Information Engineering and Computer Science, University of Trento, Italy. His current research interests focus mainly on designing secure and safe cyber-physical system of systems and socio-technical systems.

TOMMASO ZOPPI is currently a research associate with the University of Florence. He is involved in several European/National funded and even Industrial projects. He currently serves as a member of the Program Committee of several International Conferences. His research interests include focuses on anomaly detection, security and safety, often applying standards to plan, design, develop, and implement appropriate architectures or software in the domain of critical systems.

ANDREA BONDAVALLI (Member, IEEE) is a full professor of computer science with the University of Florence. His research interests include focused on dependability and resilience of critical systems and infrastructures. In particular, he has been working on designing resiliency, safety, security, and on evaluating attributes such as reliability, availability and performability. He led various national and European projects and has been chairing the PC in several International Conferences in the field, Andrea Bondavalli is a member of the IFIP W.G. 10.4 Working Group on "Dependable Computing and Fault-Tolerance".