Title

Ludovico Toscano

# Abstract

In this thesis, we will see how explainable algorithms such as Shap can play a crucial role in improving the niche of network intrusion detection systems by helping to analyze HIKARI-2021 and NFS-2023-TE. From a recent study showing how most recent NIDS works use complex models without a real need, we demonstrate how a decision tree could be better than a complex GA2Mmodel such as the explainable boosting machine. Then, thanks to Shap, we analyze different models and demonstrate why these datasets are unsuitable for real-world usage.

### Section 1 - Introduction

According to (Catillo et al., 2023) nowadays, it is easy to find papers that achieve almost perfect performance scores developing intrusion detection systems (IDS) based on machine learning (ML) models. However, according to the authors, these results do not contribute to the real world since public datasets are lacking in some areas, and the way practitioners have built these ML models has some flaws. The study breaks down these issues into two parts: the first is related to the dataset itself, and the second is about the common mistakes in building an ML model. The paper takes many times as an example CIC-IDS 2017 (Sharafaldin et al., 2017) since among the most popular datasets, it is the newest.

This work aims not only to be a practical example of what has been exposed (Catillo et al., 2023), but we will work on newer datasets and add other requirements related to building an IDS using an ML model. Moreover, we have found some things that could be improved in one of the datasets. The authors have said that future updates will be published, so we hope to have contributed to enhancing an existing dataset.

The dataset choice was NFS-2023-TE (Pekar & Jozsa, 2024) and HIKARI-2021 (Ferriyan et al., 2021). NFS-2023-TE is the latest of a series of refinements of CIC-IDS 2017, and it improves the already existing WMTC-2021 (Engelen et al., 2021) (L. Liu et al., 2022) and CRiSIS-2022 (Lanvin et al., 2023). While HIKARI-2021 is an entirely new dataset which is inspired by CIC-IDS 2017 featuring seven new content requirements on top of the 11 criteria (Gharib et al., 2016) used by the CIC-IDS 2017 authors. Since these datasets improve what has been done in previous work, we expect them to match some of the requirements expressed in (Catillo et al., 2023). Moreover, since, to the best of our knowledge, HIKARI 2021 has yet to be analyzed, we will include an analysis of the dataset in this work.

The analysis of HIKARI 2021 plays an essential role since datasets become quickly obsolete (Guerra et al., 2022), which means that practitioners must choose between obsolete but well-known datasets or ones that may contain some unknown flaws. Since it is acknowledged that using obsolete datasets should be avoided, the effort to try new datasets is necessary. This process is divided into two parts. The first will check if the same issues found on CIC-IDS 2017 affect HIKARI 2021, while the second will use eXplainable AI (XAI) algorithms for EDA. Moreover, by using XAI algorithms such as Shap (Lundberg & Lee, 2017) we will go beyond creating a model that achieves good metrics performance; we will learn about feature importance and how the model behaves.

The non-goal of this work is to make a definitive guide on making IDS based on ML models, as we will see some of the issues still need to be addressed. While trying to make a new starting point, we know this work could contain some of the abovementioned issues.

### Section 2 – Background Knowledge

This section contains some technical knowledge needed for those who are not computer scientists. Some of these explanations can be argued, but a more technical explanation is out of scope and will lead to a longer and more complex explanation.

What does an IDS do?

An IDS is usually a physical device connected to the router. Its job is to read all the connections flowing through the router and search for malicious traffic. If an IDS finds some malicious traffic, it tells the router to close all the related connections.

To be effective, an IDS needs not only to detect all the attacks but also to be as fast as possible in closing the connection with the attacker. For example, in an attack that sends all your data to the attacker's server, you want the attack stopped as soon as possible. In this case, the slower the IDS, the higher the number of stolen files.

In addition, the IDS should be computationally cheap as possible. The reason is that if the ML model is too big for the device hosting the IDS, an attacker could send many packets to the target network, making the IDS out of service. In this case, there are two options: close all the connections or disable the IDS. The first case could lead to shutting down all the company's activities using a computer network. In contrast, the network will be unprotected against another attack in the second case.

What is an IP address, and what is a port?

An IP address is an identifier inside a network, like the race number that an athlete borrows for a marathon. The same athlete will have different numbers in different races, and the same number will be assigned to different athletes in different races. This identifier is unique only during one competition. At the same time, in a local area network (LAN), a device will have its unique IP address that could change in a different network or at a different time. Since the IP address is borrowed and can change over time, it is a bad idea to use the IP address as a feature for training an ML model. As well as (when randomly assigned), a race number will not help to predict the outcome of a marathon.

A port is used to identify a specific service from a host. For example, a server can host a website and an email service. The website will be an HTTP service at port 80, while the email will be a POP service at port 110. It is essential to understand that the user can change the port, so it should not be used to train a model. While the standard for HTTP is port 80, I can use port 80 for another service, and if the model is biased about port 80 being fine, an attacker can exploit this bias. All the connections should have an IP address and a port.

In the case of these datasets, IP and port are still important, even if they are not supposed to be used. If paired with a timestamp, they can make it possible to match what we see in the dataset with what we see in the pcap file. This is necessary to eventually find issues on how the dataset was built or to make the generation of new datasets with new tools possible.

What is a flow?

If I download a file from the Internet, I open a connection from the client to the server; this is a flow. The problem related to this explanation of a flow is what to consider a flow and what multiple flows are. In the case of our dataset, these flows are based on the TCP or UDP standard. The difference between these two protocols is that a TCP connection is supposed to send some opening and closing packages, making it easy to understand when the flow begins and ends. In contrast, UDP does not have an opening and closing sequence. So, in the case of UDP, we will assume that the first package we see going from the client to the server is the opening one, while for deciding when the connection is closed, we usually use a timeout based on the latest seen packet.

Deciding on the correct threshold for this timeout is a problem without a proper answer. A longer timeout can be problematic since it can delay the identification of the threat and lead to seeing two connections as one connection. The fact that two different connections can be identified as one is rare because while the IP remains the same during the capturing session and the server port remains the same, the client port changes.

While the timeout problem is always there for UDP connections, TCP connections, for some reason, may not be closed properly. In that case, the TCP connection will be closed after a timeout.

How these datasets are made

Making a dataset for training an IDS starts with setting up a testbed to do the data capturing part, which will lead to storing the data captured in pcap files and processing them to make a csv file. All the steps we will discuss here are described in CIC-IDS 2017 and HIKARI 2021 papers.

The testbed consists of a network of devices, in our case, only computers and servers, but some datasets are specific to IoT devices. The main difference between IoT and generic datasets is the protocols used.

Once the testbed is set up, the capturing device starts to capture some session of regular traffic, which means that a computer is instructed to browse the Internet as an average person would do, and some session of attack, in which one or more hosts in the network are attacked.

The data we are talking about are IP packets containing TCP or UDP as payloads. Usually, this task is performed with tools such as tcpdump, which reads the flow of packets in the network and makes a pcap file of the data capturing session. These files contain the packet header, which includes all the packet metadata, such as destination, source, size of the packet, and a bunch of flags. Besides, the header packets also contain the payload, which is the data transported over the network.

Usually, these pcap files are released along with the dataset so that practitioners can analyze them and even create a new dataset from an existing pcap file. WTMC-2021, CRiSIS-2022, and NFS-2023-TE are all made from the pcap files of CIC-IDS 2017; their goal was not to create a new dataset from scratch but to improve an existing one.

Once we have made a pcap file, we can build our csv with a flowmeter tool, like CICFlowMeter (Lashkari et al., 2017) which was used for CIC-IDS 2017 or NFStream (Aouini & Pekar, 2022) used for NFS-2023-TE. These tools make the samples for our dataset; a sample is a flow composed of multiple packets. The samples are composed of flow statistics, like the packets' count, flow duration, timestamp, IP, port, and count of the flags.

We must label the samples once we have created a CSV file with all the connections. There are different ways to analyze the traffic, which are disclosed in the dataset papers. It is essential to know that once the attack's packet has been found, it will be labeled using the unique combination of IP, port, and timestamp. The timestamp of the flow is the same as the first packet that has started the flow.

Why do we use ML models?

Historically, IDS were working based on the signature of the payload data. Signature-based IDS make a signature of the data in the payload and then check if the signature matches a known attack in a database. The problem with this kind of solution is that they can only detect known attacks, and with the growing dimension of attack databases, ML models have been proven faster. Moreover, since traffic is ciphered nowadays, this technique is less effective. ML models do not rely on the payload; instead, they use the flow statistics the flowmeter makes. So, if a new attack is created, it is possible that an ML model can detect the attack if it behaves similarly to a known attack.

What is a FIN and RST flag?

To close a TCP connection properly, an exchange of four packets is needed; among them, there is the FIN packet. Ideally, the client sends the packet to the server, which answers with another FIN packet.

Another way to close a TCP connection is by sending an RST packet. These flags are contained in the header of the TCP packet. What is essential to know is that a TCP flow should contain two FIN packets or one RST packet.

### Section 3 – Related work

We selected six papers that use HIKARI-2021; these are the only six papers available as we are writing. The two most cited papers that use CIC-IDS 2017 and use Shap, the remaining are the five most cited papers about CIC-IDS 2017. Other than HIKARI-2021 and CIC-IDS 2017, some papers contain other datasets. In Table 1, we used some of the criteria mentioned in (Catillo et al., 2023) about the ML implications; it is a list of common issues related to ML models for IDS. We excluded the data partitioning issues since we are not building a neural network with memory or solving any other problem where the chronological order of the data is relevant. We also excluded the lack of transferability since we need better datasets to make a model that can be deployed in the real world. Moreover, we added explainability criteria for the papers that used a post hoc explainability algorithm. No one matches the updated dataset criteria; in the case of HIKARI-2021, the authors released a new data capture to improve the problem of an imbalanced dataset, but it has yet to be adopted. In the case of CIC-IDS 2017, we consider the updated version NFS-2023-TE, which is the latest of a series of refinements of CIC-IDS 2017. The dataset was published during the writing of this document, so it has yet to be adopted. All attacks refer to the fact that some papers do not use all the dataset categories; in this case, we usually expect the performance metrics to be higher. Attack-revealing features and ease of detection: In the case of HIKARI 2021, we found two papers that include the index generated by pandas as a feature that can help some models detect attacks. Since the attacks have been performed in a specific time range, with the index, a decision tree can detect that between specific ranges that are mostly attacks. While in NFS-2023-TE, there is a feature called protocol, since all the attacks have been performed over TCP, a model will be biased thinking that all the UDP connections are not attacks, but attacks over UDP exist. Unmotivated complexity refers to the fact that some papers propose a complex deep learning (DL) algorithm without making a comparison with simpler models. In the case of an IDS, lower inference time and computationally cheap models are advantages, so when a complex model is proposed, it should be compared to a simpler one, for example, a random forest (RF). A comparison with previous work is usually insufficient since we noticed that something different in data preparation or other metrics are used, making the comparison useless. Moreover, some work does a comparison without fine-tuning the classic ML model, making the comparison unfair. The latest is source code; this is the most important since if something is not mentioned or unclear on the paper, it can be found in the source code. Moreover, if needed, all the other mentioned aspects can be fixed. For example, metrics can be added if a paper does not use an F1 score with the source code.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Paper | *Avoid attack-revealing features and ease of detection* | *Avoid unmotivated complexity* | *Use of the evaluation metrics* | *Source code* | *Explainability* | *Updated dataset* | *All attacks* | *Dataset used* |
| (Kwon et al., 2023) | Yes | Yes | Yes | No | No | No | Yes | Hikari-2021 |
| (Noori et al., 2023) | Yes | Yes | Yes | No | No | No | No | Hikari-2021 |
| (Louk & Tama, 2023) | - | No | Yes | No | No | No | No | Hikari-2021 |
| (Rajak et al., 2022) | - | No | Yes | No | No | No | Yes | Hikari-2021 |
| (Fernandes & Lopes, 2022) | No | Yes | Yes | No | No | No | Yes | Hikari-2021 |
| (Fernandes et al., 2023) | No | Yes | Yes | No | No | No | Yes | Hikari-2021 |
| (Chauhan & Shah Heydari, 2020) | Yes | No | Yes | No | No \* | No | Yes | CIC-IDS 2017 |
| (Sarhan et al., 2022) | Yes | Yes | Yes | No | Yes | No | Yes | CIC-IDS 2017 |
| (Zavrak & Iskefiyeli, 2020) | Yes | Yes | No | No | No | No | Yes | CIC-IDS 2017 |
| (Kurniabudi et al., 2020) | Yes | Yes | No | No | No | No | Yes\*\* | CIC-IDS 2017 |
| (Yulianto et al., 2019) | Yes | Yes | Yes | No | No | No | No | CIC-IDS 2017 |
| (Maseer et al., 2021) | Yes | Yes | Yes | No | No | No | Yes\*\* | CIC-IDS 2017 |
| (J. Liu et al., 2021) | Yes | Yes | No | No | No | No | Yes | CIC-IDS 2017 |
| This work | Yes | Yes | Yes | Yes | Yes | Yes | Yes |  |

Table 1 - Related works \* shap was used but for feature selection \*\* they use all the features, but they group them (Panigrahi & Borah, 2018)

### Section 4 – Methodology

Dataset choice

In February 2024, the three datasets with the most citations in Google Scholars were CIC-IDS 2017 (Sharafaldin et al., 2018) with 3149 citations, NLS-KDD (Tavallaee et al., 2009) with 5131 citations, and UNSW-NB15 (Moustafa & Slay, 2015) with 2740 citations, these two are improvements based on the raw data of KDD Cup ’99 (Lee et al., 1999). We focus on two datasets derived from CIC-IDS 2017, illustrating why it may no longer be the best option.

By making CIC-IDS 2017, the authors have achieved for the first time the goal of covering all the 11 criteria given by (Gharib et al., 2016) by using CICFlowMeter (Lashkari et al., 2017). However, the analysis of the pcap files by (Engelen et al., 2021) and (L. Liu et al., 2022) revealed over 5% corruption in dataset labeling and non-compliance of CICFlowMeter with TCP connection closure standards (Brownlee et al., 1999). This led to the creation of WTMC-2021 and a patched version of CICFlowMeter. One year later, WTMC-2021 improved with the name of CRiSIS-2022 (Lanvin et al., 2023) by adding the attack port scan that was not labeled and sorting and removing the duplicates in the pcap file, again thanks to a new patch of CICFlowmeter. In 2023 (Pekar & Jozsa, 2024)noticed that even after the patching, CICFlowMeter was not closing the connection after a FIN or RST packet and was also making some missing or negative values, so they decided to make NFS-2023-TE by using NFStream (Aouini & Pekar, 2022). NFS-2023-TE creates a new flow at first FIN or RST flag like most of the flow analyzers do (Hofstede et al., 2014), while NFS-2023-nTE creates a new flow after a timeout like WTMC-2021 and CRiSIS-2022 do, this second dataset has been made only for the sake of comparison with its predecessor.

Things may change, however, since the authors of CICFlowMeter said in an email exchange that they are making a new tool written in Python with more than 50 new features for 130 features.

Hikari 2021 (Ferriyan et al., 2021) is a dataset inspired by CIC-IDS 2017 but includes seven new requirements; they also provide the pcap files, a csv file, and a pkl of the dataset. They offered 84 traffic features and two target features, one multi-categorical and the other one binary; among the 84 traffic features, most of them were inspired by the ones of CIC-IDS 2017. In this dataset, the authors have focused only on web attacks since 80% of the attacks nowadays are done on the application layer. So, they have performed three different attacks, but on the dataset, a fourth one is present since while they were analyzing the background traffic, they found a crypto miner attack. The raw data were extracted using tcpdump, while the labeling and analysis of the background traffic were done with Zeek, which also added some features that are not present in CIC-IDS 2017. These features are the IP address, source and destination ports, and uid. Since the paper's release, the dataset has received an update that improves the dataset by increasing the size of the minority classes.

We decided to build our models using NFS-2023-TE and HIKARI-2021, which, as far as we know, are the two most updated generic datasets. (Catillo et al., 2023) list some typical dataset issues; in Table 2, we will see if NFS-2023-TE and HIKARI-2021 have improved compared to CIC-IDS 2017 mentioned in the paper.

|  |  |  |
| --- | --- | --- |
| Issue | HIKARI-2021 | NFS-2023-TE |
| *Simplification of the data collection environment* | Not addressed | Not addressed |
| *Contemporaneity and effectiveness of the attack* | Hikari 2021 focuses only on encrypted traffic at the application layer, saying that 80% of attacks are done at this level. As far as we know, this is the most updated dataset available; this would be a good option until a new one emerges. | The data used was generated seven years ago. Moreover, some attacks in the dataset have been proven ineffective nowadays. In (Catillo et al., 2021) they |
| *Representativeness of the normal baselines* | Not addressed | Not addressed |
| *Bugs of the feature extractor and incorrect ﬂow records* | Hikari 2021 does not provide any source code and mentions that the labeling was made with Zeek alongside an undefined Python tool, so we analyzed the dataset before using it. This part will be discussed later. However, in an email exchange with the authors, they said the source code would be released. | To avoid this problem, NFS-2023-TE was made with NFStream, which was chosen since it is an open-source tool with a broad user base, and the labeling process has been documented and released alongside the Python code used. |
| *Data Labeling (Was the traffic analyzed or labeled based on IP, port, and timestamp?)* | The background data were analyzed with Zeek, which led to the discovery of an attack. HIKARI 2021 does not provide the timestamp, and part of the payload is encrypted to ensure privacy; it is not possible to prove if there is any flaw in the labeling. | NFS-2023-TE should deliver on this point since it has been made on top of the refinement that has improved the original issues of the labeling part. |
| *Class imbalance* | Probing is around 2,9 times bigger than Bruteforce, which has 7988 samples. This difference is considerably smaller than NFS-2023-TE, making over-sampling possible if needed. | PortScan is around 14574 times bigger than Hearthbleed, with only 11 samples. With such a small sample, it is impossible to make a good model. |

Table 2- Typical dataset issues

Explainability algorithms

Shap (Lundberg & Lee, 2017) is a Python library that uses a game theoretic approach by computing an approximation of the Shapley values to create post hoc explanations for any machine learning models. With this work, among other explainers, they introduced Kernel Shap, a model-agnostic explainer adapted from Lime (Ribeiro et al., 2016), and Deep Shap, a model-specific explainer for neural networks that has adapted the algorithm of Deep Lift (Shrikumar et al., 2017).

Lately, two other papers (Lundberg et al., 2018, 2020) introduced Tree Shap, a model-specific explainer for trees and ensembles of trees, which is so fast compared to Deep Shap and Kernel Shap that, in most cases, can explain the entire training set in a few seconds. However, the library Fast Tree Shap (Yang, 2021) introduced an optimized version of the Tree Shap algorithm, which is faster in single-core but can overcome the original algorithm's single-core limitation, allowing even faster parallel execution. Since Fast Tree Shap has all the characteristics of Tree Shap, when we talk about Tree Shap, we will refer to both Tree Shap and Fast Tree Shap algorithms.

Tree Shap explainers and Linear Shap can provide true-to-the-model explanations (H. Chen et al., 2020) instead of other explainers that are true to the data. By breaking the dependence between features by following the rules of causal inference (Janzing, 2019), true-to-the-model explanations assign a value different from zero to a feature only if the model uses it. In the case of a true-to-the-data explanation, if two features correlate 100%, the same importance will be assigned to them even if one is not used.

Even though Tree Shap is a fast post hoc explainer, a glass box model like the Explainable Boosting Machine (EBM) (Lou et al., 2013) does not require any post hoc explanation. Being a GA2M model, it promises to be accurate as a state-of-the-art black box model without sacrificing explainability.

ML model choice

In the survey (Zhang et al., 2022), the authors highlight some limitations of explainable AI for cyber security, a big subject that includes IDS. We will focus on how two future improvements highlighted in this survey. The first is a high-quality dataset, which has been covered in this work. The second future improvement mentioned is the trade-off between explainability and performance. Decision tree (DT) was one of the chosen models that, with the addition of tree shap, helped to understand the model itself and how these datasets are made. Then we have boosted models paired with tree shap; the ability of tree shap to provide a true-to-the-model explanation with the incredible performance of these models has driven this choice. Boosted models can handle imbalanced data very well; in the case of NFS-2023-TE, this is a requirement given the small size of some categories. We have chosen Random Forest (Breiman, 2001), LightGBT (Ke et al., 2017), XGBoost (T. Chen & Guestrin, 2016), CatBoost (Dorogush et al., 2017) and EBM.

Preprocessing

To achieve better results, we opted to under-sampling all the datasets. For HIKARI-2021, all the classes were sampled to have 7988 samples for each class; moreover, we merged Bening and background traffic. This number has been chosen since the smaller class is that big. Meanwhile, for NFS-2O23-TE, under-sampling was performed to have 738 samples for each class except for smaller classes. In this case, we decided to test different numbers of samples, and this was the one performing the best. This is a significant limitation of this study, but the alternative of removing some attacks is not that better. With only 11 samples in the smallest class, over-sampling was not considered to avoid overfitting the synthetic data.

We used 80% of the stratified data for the training split to keep the same ratio between classes. Moreover, we performed 5-fold cross-validation.

Evaluation metrics

We used two metrics available in sklearn called F1 macro and F1 weighted.

Where N is the number of classes, i is a class, and wi is the weight of a class in the entire dataset. We can notice that in the case of a balanced dataset there is no difference between the two equations, that’s the case of HIKARI 2021 after the under sampling.

Section 5 – Results

Data set analysis

While in NFS-2023-TE, we did not find any flaw in the dataset, in HIKARI 2021, we noticed that the flag count for FIN and RST is higher than how it is supposed to be, having a maximum of 140 FIN packages and 110 RST packages. Moreover, the most extended flow lasted 17942 seconds, about 4.9 hours. Since the capturing sessions of HIKARI 2021 lasted 3 to 5 hours, the tool did not have a timeout, nor was it closing the connection at the first RST or FIN flag. Likely, the tool was always ignoring any timeout, RST, or FIN packet since if we filter for all the connections that lasted less than 181 seconds, with less than 2 RST and 3 FIN packets, we obtain only 104 samples for bruteforce and 59 for bruteforce-XML. For instance, NFS-2023-TE closes a connection after 120 seconds, and CICFlowMeter was supposed to close a connection after the 2 FIN packets. This evidence led to a possible conclusion that the flows within the range we selected were those with a unique IP and port combination during the capture. If two or more connections open with the same combination during the capture, they show these unexpected statistics.

Since there is no documentation about the labeling and the authors still need to release the timestamps, we cannot make it suitable for real-world usage. Instead, we decided to show how explainability can be used to discover these flaws.

We also analyzed the Pcap files using the same tools as the CRiSIS 2022 reordercap and editcap with a window of 500 microseconds. We found that 0,03% of the packages were duplicated, and 0,23% were out of order. The authors confirmed that issue and did not perform preprocessing before making the dataset.

Model performance

The following training, classification, and explanation times have been measured on a Dell XPS 13 9315 with a 12th gen i7-1250u and 16gb of ddr5 ram running Fedora 39 with Linux 6.8. For the environment information, there is a YAML file on the GitHub repository with all the versions of the packages in the conda environment. It is crucial to notice that Sklearnex and daal4py have been used to improve inference time, and all the models have been set to use all the cores available.

The parameter tuning of Table 3 was done following the suggestions of the documentation for each library and by looking at what was working for the other models. We used the metrics f1 macro of sklearn to evaluate the results, and when different parameters were giving similar results, a shorter training time was used to lead the decision.



Table 3 - NFS-2023-TE and NFS-2023-nTe parameters

Table 4 shows the training time, explanation time, and the sum of both in ms for each model tested with 6087 samples. Fast tree shap is set to choose the best algorithm for the explanation automatically. These times have some variance but show the order of magnitude necessary to run each model. We can notice that even if EBM is the slowest during training, it can be faster than lightgbm and xgboost if we consider the explanation time. The random forest is close to the decision tree thanks to using only ten trees, which is considerably less than the GAM models. Another thing to consider is that even if Catboost uses fewer trees than the lightgbm and xgboost, it is slower to train.



Table 4 - NFS-2023-TE training times

In Table 5, we compare the prediction time in μs for 1522 samples; in this case, we have used the magic function %timeit of ipython to compute the mean of different runs. Doing the mean is essential because it is statistically more significant. With such a smaller duration, a warmup of the function is necessary to reduce the computation time. Without the warmup run, the classification time of the decision tree was 3 ms because, at the first run, the computer loaded the prediction function into memory, which took more time than the prediction itself.



Table 5- NFS-2023-TE – average F1 score of 5-fold

Looking at the EBM, we notice that it is the best with F1 macro but the worst with F1 weighted. Comparing these results with the decision tree shows that the decision tree has a problem with Hearthbleed and Web Attack - SQL Injection, which misclassifies two samples for the first and 4 with the latter. In comparison, the EBM misclassifies 111 samples in the DoS Slowhttptest. These results expose the need for a balanced dataset; we can argue that EBM is the best one because it can handle each attack well, but on the other hand, building a model over 11 samples will lead to something that will not work in the real world.

Moreover, we need to consider the prediction time; by looking at the F1 results, lightgbm is probably the winner by being the second-best in both macro and weighted scores but the worst in prediction time. Lightgbm is twice as slow as the random forest, which scores nearly the same in the F1 metrics, making the random forest the best alternative overall. The classification time is so important because, in case of a DoS attack that opens and closes the connection at a fast enough speed, the model needs to keep up with each new flow generated; otherwise, the NIDS will run out of service. Another reason why this classification time is necessary is that it can make a difference between enabling or not the use on the edge and lead to less expensive devices when the GPU is not required for running an NIDS.



Table 6 - HIKARI 2021 parameters

Table 6 shows how the parameters for HIKARI-2021 lead to less complex models except for the ebm. One of the reasons is that this dataset is balanced after the under-sampling.



Table 7- HIKARI-2021 training times

Therefore, Table 7 shows lower training times for the 31952 training samples compared to NFS-2023-TE. EBM is the only model that has increased the training time because, contrary to the other models, it was impossible to build a smaller model with fine-tuning.



Table 8 - HIKARI-2021 F1 score and times

Table 8 shows how the model performs nearly the same regarding the F1 score, while the faster model was the decision tree, making it the best model for HIKARI-2021.

Model explanations

We decided to compare the explanations of the decision tree with the one of the catboost since they decided to give different features importance. Moreover, we printed the decision tree to analyze why tree shap gave more importance to some features.

Figure 1 shows the feature importance of a decision tree for the class bot of NFS-2023-TE. The model has misclassified only 1 sample out of 737 using only a small subset of the available features. The same pattern of using a few features applies to the other attacks. The only exception is benign traffic, which requires more features to be detected. The benign class assigns positive importance to more features because most of the split leads to either attack or benign traffic.

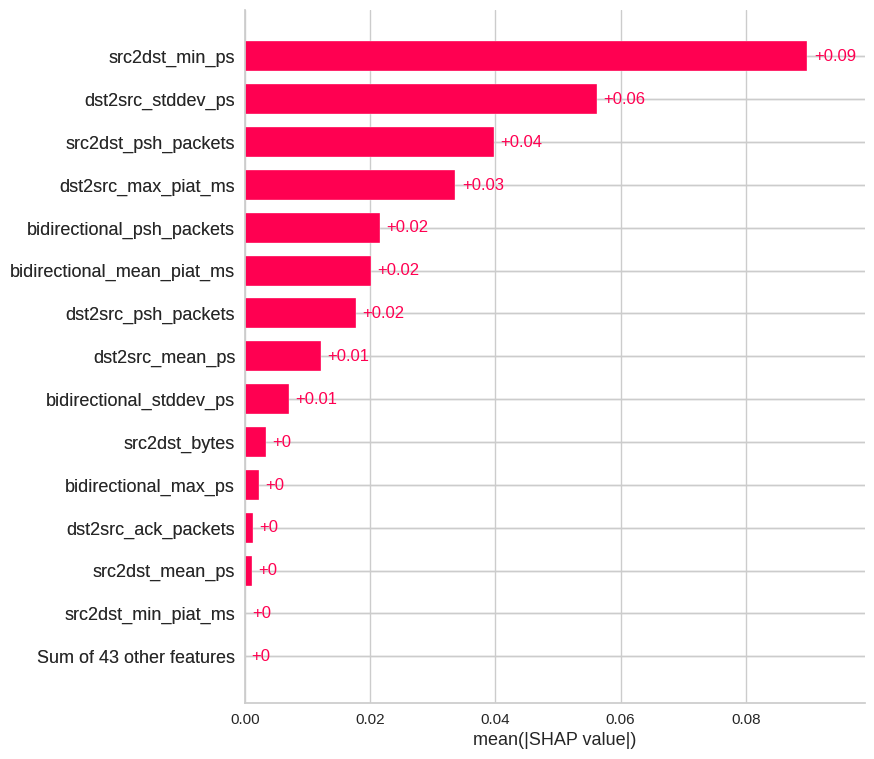


Figure 1 - NFS-2023-TE - decision tree feature importance with shap of bot class

Immagine che contiene testo, schermata, numero, linea

Descrizione generata automaticamenteFigure 2 - HIKARI-2021 - decision tree feature importance with shap of bruteforce-XML class

In Figure 2, we can see that the decision tree has given almost all the importance to one feature; with only the first two nodes, the decision tree can classify bruteforce-XML with a gini coefficient of 0.041. This means that 6388 of the 6526 were correctly classified.

Table 9 shows the quintile grouped by category for the first feature shown in Figure 2, knowing that the root node of the tree selects all the samples less or equal to 737 and that the false branch filters all the samples less or equal to 761.5 made clear why this feature is the most important detecting a bruteforce-XML attack. The problem is that this quintile distribution of this feature is not valid in the real world. Moreover, changing the payload size can fool a model built with these data.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **category** | **min** | **25%** | **50%** | **75%** | **max** |
| **Background** | 0 | 0 | 40 | 127 | 15741 |
| **Benign** | 0 | 0 | 36 | 426 | 3456 |
| **Bruteforce** | 0 | 357 | 373 | 425 | 786 |
| **Bruteforce-XML** | 0 | 746 | 747 | 748 | 925 |
| **Probing** | 517 | 517 | 517 | 517 | 517 |
| **XMRIGCC CryptoMiner** | 40 | 40 | 50 | 50 | 232 |

Table 9 - fwd\_pkts\_payload.max quintiles

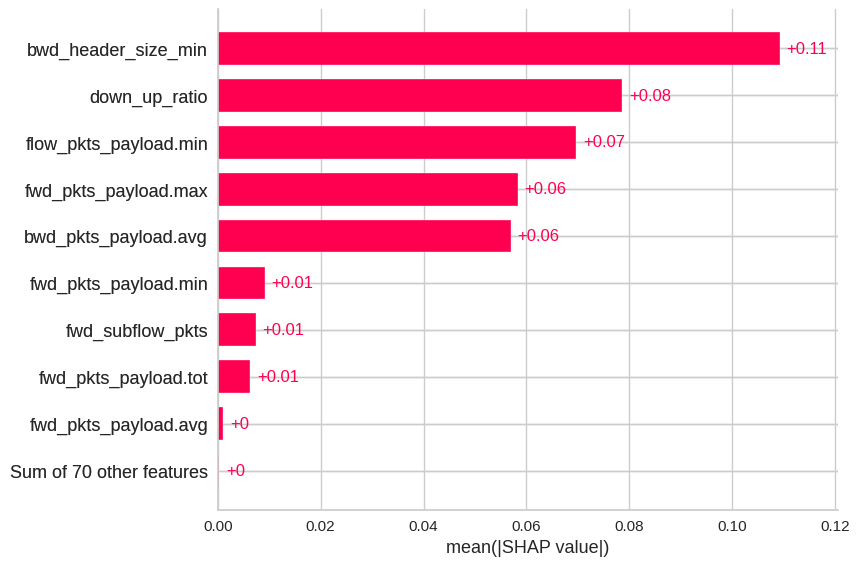


Figure 3- HIKARI-2021 - decision tree feature importance with shap of XMRIGCC CryptoMiner

Figure 3 shows the importance of the features of XMRIGCC CryptoMiner with the decision tree. Again, only a few selected features have been used; the most important is bwd\_header\_size\_min which for this attack is always zero then we have down\_up\_ratio. Table 10 shows that the crypto miner is the only one with always 0 for this feature, making it easy to understand why it is so important for this class. Down\_up\_ratio is also the most important feature for probing, where we can see that the mean value is around 1.3, with a low standard deviation compared to the other attacks. This is the second most important feature for classifying background, but in this case, we do not have any hypothesis why.

|  |  |  |
| --- | --- | --- |
| **category** | **mean** | **std** |
| **Background** | 0.810244 | 0.721923 |
| **Benign** | 2.015250 | 32.321667 |
| **Bruteforce** | 8.794107 | 89.440773 |
| **Bruteforce-XML** | 20.071672 | 128.065872 |
| **Probing** | 1.298040 | 0.119266 |
| **XMRIGCC CryptoMiner** | 0.000000 | 0.000000 |

Table 10 - Down\_up\_ration mean and std

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamenteFigure 4 - NFS-2023-TE - catboost impact on the model output of DDoS with shap sorted by highest magnitude of impact

Figure 4 is the explanation of the catboost algorithm where we can see that the feature with the highest magnitude of all is bidirectional\_fin\_packets, which shows up for Bot, DDoS, DoS Slowhttptest, DoS Slowloris, Heartbleed, Infiltration, and Web Attack – XSS as one of the two most important feature in average or the one with the highest magnitude of impact. By analyzing this feature, we can notice that Bot and Web Attack -XSS always have 1 FIN packet, while DDoS averages 99% of the samples with 1 FIN packet. This characteristic makes the difference in the other attacks cited at the 85th percentile of the distribution showing 0 FIN packets. To understand why we have a lot of 0 we analyzed the duration and for Infiltration and Heartbleed, most of the connection gets closed because of the timeout, while for DoS Slowhttptest and DoS Slowloris, it is hard to tell since they did not reach the timeout either they send an RST packet.

All these analyses conclude that using a more complex model than a decision tree or a random forest is difficult to justify for these datasets. However, this does not mean that the decision tree is ready for deployment in the real world. The reason behind this algorithm's success is the oversimplified test bed.

Section 6 – Conclusion

We first saw that CICIDS-2017 is nowadays one of the most popular datasets cited in Google Scholar, and then we saw how new proposals such as NFS-2023-TE and HIKARI-2021 aim to improve the work done with CICIDS-2017. In particular, NFS-2023-TE uses the same raw data as CICIDS-2017, while HIKARI-2021 proposes a new, updated test bed.

Then, we analyzed the flaws in the current literature and the dataset in use, which led to the proposal of a new methodology, with the addition of the explainability part, as seen in other work.

HIKARI-2021 has been analyzed using the criteria proposed by NFS-2023-TE, which led to the discovery of some flaws in the raw data and the dataset generation.

Finally, the analysis of the results shows how comparing simpler models such as the decision tree is necessary to have a benchmark for more complex models. Moreover, we have shown how explainable AI algorithms such as Shap need to be used to help build better models and to increase the accountability of the proposed models. Thanks to the explanations, we discovered something about the dataset, which has helped us avoid mistakes during the work. More than once, features that were not supposed to be in the training set were identified thanks to Shap.

Aggiungere che HIKARI può migliorare dopo aver sistemato il bug che non chiude mai le connessioni, però resta il fatto che comunque il test bed è troppo semplice.

Aouini, Z., & Pekar, A. (2022). NFStream: A flexible network data analysis framework. *Computer Networks*, *204*. https://doi.org/10.1016/j.comnet.2021.108719

Breiman, L. (2001). Random forests. In *Random Forests* (Vol. 45). https://doi.org/10.4324/9781003109396-5

Brownlee, N., Mills, C., & Ruth, G. (1999). *Traffic Flow Measurement: Architecture*.

Catillo, M., Pecchia, A., Rak, M., & Villano, U. (2021). Demystifying the role of public intrusion datasets: A replication study of DoS network traffic data. *Computers & Security*, *108*, 102341. https://doi.org/10.1016/j.cose.2021.102341

Catillo, M., Pecchia, A., & Villano, U. (2023). Machine Learning on Public Intrusion Datasets: Academic Hype or Concrete Advances in NIDS? *Proceedings - 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume, DSN-S 2023*, 132–136. https://doi.org/10.1109/DSN-S58398.2023.00038

Chauhan, R., & Shah Heydari, S. (2020). *Polymorphic Adversarial DDoS attack on IDS using GAN*.

Chen, H., Janizek, J. D., Lundberg, S., & Lee, S.-I. (2020). *True to the Model or True to the Data?* https://github.com/slundberg/shap/blob/

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *13-17-August-2016*, 785–794. https://doi.org/10.1145/2939672.2939785

D’hooge, L., Verkerken, M., Wauters, T., De Turck, F., & Volckaert, B. (2023). Castles Built on Sand: Observations from Classifying Academic Cybersecurity Datasets with Minimalist Methods. *International Conference on Internet of Things, Big Data and Security, IoTBDS - Proceedings*, *2023-April*, 61–72. https://doi.org/10.5220/0011853300003482

Dorogush, A. V., Ershov, V., & Yandex, A. G. (2017). *CatBoost: gradient boosting with categorical features support*. https://github.com/Microsoft/LightGBM

Engelen, G., Rimmer, V., & Joosen, W. (2021). Troubleshooting an Intrusion Detection Dataset: The CICIDS2017 Case Study. *Proceedings - 2021 IEEE Symposium on Security and Privacy Workshops, SPW 2021*, 7–12. https://doi.org/10.1109/SPW53761.2021.00009

Fernandes, R., & Lopes, N. (2022). Network Intrusion Detection Packet Classification with the HIKARI-2021 Dataset: a study on ML Algorithms. *10th International Symposium on Digital Forensics and Security, ISDFS 2022*, 1–5. https://doi.org/10.1109/ISDFS55398.2022.9800807

Fernandes, R., Silva, J., Ribeiro, O., Portela, I., & Lopes, N. (2023). The impact of identifiable features in ML Classification algorithms with the HIKARI-2021 Dataset. *ISDFS 2023 - 11th International Symposium on Digital Forensics and Security*, 1–5. https://doi.org/10.1109/ISDFS58141.2023.10131864

Ferriyan, A., Thamrin, A. H., Takeda, K., & Murai, J. (2021). Generating network intrusion detection dataset based on real and encrypted synthetic attack traffic. *Applied Sciences (Switzerland)*, *11*(17). https://doi.org/10.3390/app11177868

Gharib, A., Sharafaldin, I., Habibi, A., & Ghorbani, A. (2016). *An Evaluation Framework for Intrusion Detection Dataset*.

Guerra, J. L., Catania, C., & Veas, E. (2022). Datasets are not enough: Challenges in labeling network traffic. *Computers & Security*, *120*, 102810. https://doi.org/10.1016/j.cose.2022.102810

Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., & Pras, A. (2014). Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX. *IEEE Communications Surveys and Tutorials*, *16*(4), 2037–2064. https://doi.org/10.1109/COMST.2014.2321898

Janzing, D. (2019). *Causal Regularization*.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. *31st Conference on Neural Information Processing Systems*. https://github.com/Microsoft/LightGBM.

Kurniabudi, Stiawan, D., Darmawijoyo, Bin Idris, M. Y. Bin, Bamhdi, A. M., & Budiarto, R. (2020). CICIDS-2017 Dataset Feature Analysis with Information Gain for Anomaly Detection. *IEEE Access*, *8*, 132911–132921. https://doi.org/10.1109/ACCESS.2020.3009843

Kwon, D., Neagu, R. M., Rasakonda, P., Ryu, J. T., & Kim, J. (2023). Evaluating Unbalanced Network Data for Attack Detection. *SNTA 2023 - Proceedings of the 2023 on Systems and Network Telemetry and Analytics*, 23–26. https://doi.org/10.1145/3589012.3594898

Lanvin, M., Gimenez, P. F., Han, Y., Majorczyk, F., Mé, L., & Totel, É. (2023). Errors in the CICIDS2017 Dataset and the Significant Differences in Detection Performances It Makes. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *13857 LNCS*, 18–33. https://doi.org/10.1007/978-3-031-31108-6\_2

Lashkari, A. H., Gil, G. D., Mamun, M. S. I., & Ghorbani, A. A. (2017). Characterization of tor traffic using time based features. *ICISSP 2017 - Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, *2017-Janua*, 253–262. https://doi.org/10.5220/0006105602530262

Lee, W., Stolfo, S. J., & Mok, K. W. (1999). *Mining in a Data-flow Environment: Experience in Network Intrusion Detection*.

Liu, J., Gao, Y., & Hu, F. (2021). A fast network intrusion detection system using adaptive synthetic oversampling and LightGBM. *Computers and Security*, *106*, 102289. https://doi.org/10.1016/j.cose.2021.102289

Liu, L., Engelen, G., Lynar, T., Essam, D., & Joosen, W. (2022). Error Prevalence in NIDS datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018. *2022 IEEE Conference on Communications and Network Security, CNS 2022*, 254–263. https://doi.org/10.1109/CNS56114.2022.9947235

Lou, Y., Caruana, R., Gehrke, J., & Hooker, G. (2013). Accurate intelligible models with pairwise interactions. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *Part F1288*, 623–631. https://doi.org/10.1145/2487575.2487579

Louk, M. H. L., & Tama, B. A. (2023). Dual-IDS: A bagging-based gradient boosting decision tree model for network anomaly intrusion detection system. *Expert Systems with Applications*, *213*, 119030. https://doi.org/10.1016/j.eswa.2022.119030

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., & Lee, S. I. (2020). From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, *2*(1), 56–67. https://doi.org/10.1038/s42256-019-0138-9

Lundberg, S. M., Erion, G. G., & Lee, S.-I. (2018). *Consistent Individualized Feature Attribution for Tree Ensembles*. http://github.com/slundberg/shap

Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, *2017-Decem*, 4766–4775. https://github.com/slundberg/shap

Maseer, Z. K., Yusof, R., Bahaman, N., Mostafa, S. A., & Foozy, C. F. M. (2021). Benchmarking of Machine Learning for Anomaly Based Intrusion Detection Systems in the CICIDS2017 Dataset. *IEEE Access*, *9*, 22351–22370. https://doi.org/10.1109/ACCESS.2021.3056614

Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *2015 Military Communications and Information Systems Conference, MilCIS 2015 - Proceedings*, 1–6. https://doi.org/10.1109/MilCIS.2015.7348942

Noori, M. S., Sahbudin, R. K. Z., Sali, A., & Hashim, F. (2023). Feature Drift Aware for Intrusion Detection System using Developed Variable Length Particle Swarm Optimization in Data Stream. *IEEE Access*, *11*(September), 1–1. https://doi.org/10.1109/access.2023.3333000

Panigrahi, R., & Borah, S. (2018). A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems. *International Journal of Engineering and Technology(UAE)*, *7*(3.24 Special Issue 24), 479–482. https://www.researchgate.net/publication/329045441

Pekar, A., & Jozsa, R. (2024). *Evaluating ML-Based Anomaly Detection Across Datasets of Varied Integrity: A Case Study*. https://intrusion-detection.distrinet-research.be/WTMC2021/tools

Rajak, P., Lachure, J., & Doriya, R. (2022). CNN-LSTM-based IDS on Precision Farming for IIoT data. *Proceedings of 4th International Conference on Cybernetics, Cognition and Machine Learning Applications, ICCCMLA 2022*, 99–103. https://doi.org/10.1109/ICCCMLA56841.2022.9988997

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). *“Why Should I Trust You?” Explaining the Predictions of Any Classifier*. https://doi.org/10.1145/2939672.2939778

Sarhan, M., Layeghy, S., & Portmann, M. (2022). Evaluating Standard Feature Sets Towards Increased Generalisability and Explainability of ML-Based Network Intrusion Detection. *Big Data Research*, *30*, 100359. https://doi.org/10.1016/j.bdr.2022.100359

Sharafaldin, I., Gharib, A., Lashkari, A. H., & Ghorbani, A. A. (2017). Towards a Reliable Intrusion Detection Benchmark Dataset. *Software Networking*, *2017*(1), 177–200. https://doi.org/10.13052/jsn2445-9739.2017.009

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy*, *2018-Janua*(Cic), 108–116. https://doi.org/10.5220/0006639801080116

Shrikumar, A., Greenside, P., Shcherbina, A. Y., & Kundaje, A. (2017). *Not Just A Black Box: Learning Important Features Through Propagating Activation Differences*. https://arxiv.org/

Slack, D., Hilgard, S., Jia, E., Singh, S., & Lakkaraju, H. (2020). Fooling LIME and SHAP: Adversarial attacks on post hoc explanation methods. *AIES 2020 - Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, 180–186. https://doi.org/10.1145/3375627.3375830

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009*, *Cisda*, 1–6. https://doi.org/10.1109/CISDA.2009.5356528

Yang, J. (2021). *Fast TreeSHAP: Accelerating SHAP Value Computation for Trees*. http://arxiv.org/abs/2109.09847

Yulianto, A., Sukarno, P., & Suwastika, N. A. (2019). Improving AdaBoost-based Intrusion Detection System (IDS) Performance on CIC IDS 2017 Dataset. *Journal of Physics: Conference Series*, *1192*(1). https://doi.org/10.1088/1742-6596/1192/1/012018

Zavrak, S., & Iskefiyeli, M. (2020). Anomaly-Based Intrusion Detection from Network Flow Features Using Variational Autoencoder. *IEEE Access*, *8*, 108346–108358. https://doi.org/10.1109/ACCESS.2020.3001350

Zhang, Z., Hamadi, H. Al, Damiani, E., Yeun, C. Y., & Taher, F. (2022). Explainable Artificial Intelligence Applications in Cyber Security: State-of-the-Art in Research. *IEEE Access*, *10*(July), 93104–93139. https://doi.org/10.1109/ACCESS.2022.3204051