Future Generation Computer Systems 160 (2024) 951–965

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Anomaly-based error and intrusion detection in tabular data: No DNN outperforms tree-based classifiers

Tommaso Zoppi [a,*], Stefano Gazzini [b], Andrea Ceccarelli [b]

[a] Department of Information Engineering & Computer Science, University of Trento, Via Sommarive 9, Povo (Trento), Italy
[b] Department of Mathematics and Informatics, University of Florence, Viale Morgagni 65, Florence, 50134, Florence, Italy

ARTICLE INFO

Keywords:
Tabular data
Classification
Error detection
Anomaly detection
Intrusion detection
Ensembles

ABSTRACT

Recent years have seen a growing involvement of researchers and practitioners in crafting Deep Neural Networks (DNNs) that seem to outperform existing machine learning approaches for solving classification problems as anomaly-based error and intrusion detection. Undoubtedly, classifiers may be very diverse among themselves, and choosing one or another is typically due to the specific task and target system. Designing and training the optimal tabular data classifier requires extensive experimentation, sensitivity analyses, big datasets, and domain-specific knowledge that may not be available at will or considered a non-strategical asset by many companies and stakeholders. This paper compares, using a total of 23 public datasets: i) traditional (tree-based, statistical) supervised classifiers, ii) DNNs that are specifically designed for classifying tabular data, iii) DNNs for image classification that are applied to tabular data after converting data points into images, alone and as ensembles. Experimental results and related discussions show clear advantages in adopting tree-based classifiers for anomaly-based error and intrusion detection in tabular data as they outperform their competitors, including DNNs. Then, individual classifiers are compared against ensembles using different combinations of the classifiers considered in this study as base-learners, providing a unified final response through many meta-learning strategies. Results show that there is no benefit in building ensembles instead of using a tree-based classifier as Random Forests, eXtreme Gradient Boosting or Extra Trees. The paper concludes that anomaly-based error and intrusion detectors for critical systems should use the old (but gold) tree-based classifiers, which are also easier to fine-tune, and understand; plus, they require less time and resources to learn their model.

1. Introduction

Nowadays the paradigm of Cyber-Physical Systems (CPSs [1]) guides the definition and design of ICT hardware-software systems whose functionalities are partially controlled or monitored by computer-based sub-systems and/or human beings. Examples include, but are not limited to, Auto-Pilot Avionics, Autonomous Driving, Smart Manufacturing, Medical Support Systems, Industrial Control Systems, and Environmental Monitoring [2–6]. Noticeably, many of those CPSs (systems from now on) might be intended to deliver critical functionalities, whose malfunction may lead to fatalities, severe injuries, or major damages to the environment: as a result, they must be conceptualized, designed, and implemented to ensure that appropriate safety and/or security requirements are met [7,8]. These critical systems need to embed error, intrusion, and anomaly detectors that can accurately and promptly detect the manifestation of faults or attacks (i.e., anomalies), before subsequent cascading effects could create a major damage the encompassing system. Detectors process tabular data points containing values of specific indicators monitored from the target system (e.g., resource usage, active threads, application-specific indicators): once anomalies are detected, they trigger reaction strategies that break the fault-error-failure chain and ultimately block the system from failing uncontrollably [7].

The analysis of monitored data is conducted using Machine Learning (ML) algorithms that perform binary classification (classifiers from now on) and as such can classify the system's behavior as normal or anomalous due to errors or attacks that already happened or are currently happening in the system. ML algorithms that rely on Decision Trees or tree ensembles (Random Forests, eXtreme Gradient Boosting) were traditionally used for classifying tabular data as they build accurate models, require limited training and test time, and can be explained fairly easily [9,10]. However, there is a growing interest in crafting Deep

* Corresponding author.
E-mail address: tommaso.zoppi@unitn.it (T. Zoppi).

https://doi.org/10.1016/j.future.2024.06.051
Received 11 January 2024; Received in revised form 3 June 2024; Accepted 22 June 2024
Available online 29 June 2024

Neural Networks (DNNs) that have excellent classification capabilities when dealing with tabular data, e.g., TabNet [11], NODE [12], and FastAI [13]. These algorithms learn complex models that are usually adequate for extracting knowledge from big datasets [14–16]. This led to comparison and benchmarking studies that show how DNNs outperform tree-based classifiers [11,12], or vice-versa [9,17]: the community did not reach an agreement yet as different studies seem to suggest different outcomes. This is very unfortunate as clear directions could rapidly become best practices to speed up any study that aims at crafting optimal tabular data classifiers with immediate research and industrial relevance.

To solve this debate, we exercise i) traditional supervised classifiers (including Decision Tree classifiers and tree-based ensembles), ii) DNNs is specifically designed for the analysis of tabulr data, and iii) DNNs for image classification that we exercise after reshaping tabular data points as images [18,19] on different scenarios related to anomaly-based error and intrusion detection, collecting results and conducting in-depth analyses and discussions. Then, we use them as base-learners to build ensembles, which are typically considered a promising option to improve classification performance [20–23]. The final experimental study using 23 public datasets related to error and intrusion detection, relying on 18 classifiers, computing diversity (i.e., disagreement between base-learners), and classification metrics allows drawing conclusive statements that are valid for anomaly-based error and intrusion detectors but may also translate to similar domains. Our takeovers, supported by experimental results, are as follows.

- DNNs never outperform tree-based supervised classifiers in intrusion, error or anomaly datasets used in this study.
- DNNs have many hyperparameters that should be assigned after fine-tuning through tailored and algorithm-specific sensitivity analyses, which are very time-consuming and require domain-specific knowledge.
- DNNs do not outperform tree-based ensembles even when processing "big data" tabular training datasets
- DNNs take at least an order of magnitude (10x) more time for learning a model than tree-based ensembles. Tabular DNNs are on average faster than image DNNs trained on images converted from tabular data points.
- Choosing base-learners that are likely to disagree, resulting in diverse behavior, does not help in building ensembles that outperform individual anomaly-based detectors.
- There is no benefit in using tabular classifiers other than tree-based ensembles for anomaly-based error and intrusion detection. This is true for classification performance of classifiers alone, and also when crafting complex classifiers using ensembles.

The paper is structured as follows: Section 2 recaps error and intrusion detectors and the role of ML in those domains, including a discussion of ensembles of classifiers, their diversity, and ways to combine them through meta-learning. Section 3 shows motivations of this study, while Section 4 conducts a first experimental analysis to compare classifiers individually. Section 5 describes ensembles of tree-based classifiers and DNNs, which are used for experiments and discussions in Section 6. Section 7 summarizes threats to validity, and Section 8 concludes the paper.

2. Background

This section reports the background on tabular datasets and Machine Learning (ML) algorithms that suit the classification of such data i.e., classifiers.

2.1. Anomaly-based error and intrusion detection

Dependability is generally referred to as "*the ability to avoid service failures that are more frequent or severe than is acceptable*" [7]. Attaining dependability requires - but is not limited to - a prompt detection of the observable manifestations of faults or attacks, which should trigger reaction strategies to avoid uncontrolled system failures. Error [15,16,24] and intrusion [25–27] detectors are classifiers that aim at detecting all the manifestations of faults (error detection) or attacks (intrusion detection). They seek to distinguish between normal behavior and one or more anomalous categories of anomalous behaviors due to manifestations of errors or intrusions. These manifestations usually occur as behavioral anomalies, which are observable when looking at specific performance indicators. Detectors may occasionally fail, either by triggering unnecessary alerts (False Positives, FPs), or when they miss the detection of an ongoing fault or attack (False Negatives, FNs). Usually, error and intrusion detectors primarily focus on reducing FNs, which may have a direct detrimental impact on a system. On the other hand, a very suspicious detector that has very low FNs at the price of increasing FPs will likely raise many false alarms, being of no practical use. Crafting error and intrusion detectors that output a satisfactorily low amount of FPs and FNs is not trivial, and heavily depends on two key tasks: i) precise monitoring of the target system, and ii) a suitable data analysis strategy.

2.2. Monitoring and tabular datasets

Over the years, research and practice have devised different ways to install monitoring probes into a system. Those probes aim at retrieving the value of several performance indicators of the target system at a given instant, averaged over a time frame, or signaled when specific events occur. The results of monitoring activities [3,4] constitute a structured tabular data baseline. Different performance indicators, or system features, can be targeted depending on the specific task, ranging from hardware or low-level [3], system-level [28], environment [4], or application-level monitoring [29]. Noticeably, features should describe the behavior of the system without being affected by the specific setup of an experimental campaign. As a specific case, IP Addresses should be disregarded when training intrusion detectors, since we can hardly assume to know the IP address of the attacker(s).

The resulting tabular dataset has specific properties compared to other tabular datasets. Particularly, features can hardly be considered independent as they describe different viewpoints of the same system or different areas of the same system. This may become a problem whenever applying classifiers that are known to perform well under the assumption of (linear) independence amongst features. Moreover, anomaly-based error and intrusion detection datasets for critical systems are usually collected by exercising a monitoring system over a quite stretched timespan: thus, they will have many data points but not as many features, which hardly exceed hundreds. Monitoring thousands of features every time may be possible, but it will critically slow down the execution of the regular tasks of the system, which should not be negatively impacted by monitoring and logging activities.

2.3. Classification of tabular data

A tabular dataset can be analyzed for different purposes, either to learn optimization processes [30,31] or to learn models that can be used for predicting properties of unseen data points. Within this paper, tabular data is meant to be provided to ML algorithms, which will use it to learn how to classify normal against anomalous system behavior, and ultimately detect errors or intrusions through binary or multi-class classification. The vast majority of ML algorithms that have been used for decades to tackle classification tasks are supervised classifiers [20,32–34]. Those classifiers require training data for which the label (also called class) is known. Depending on the way they learn their model, supervised classifiers are usually partitioned into tree-based classifiers (mostly Decision Trees to build ensembles as Random Forests [35,36] or XGBoost [37]), statistical techniques [38], distance-based learners [39], or neural networks (DNNs, [32,40]).

DNNs are supervised classifiers that contain multiple hidden layers (deep networks) to learn different features with multiple levels of abstraction [41]. Those classifiers learn complex representations of features during training, creating a neural network composed of multiple layers that build upon such increasingly informative features. This guarantees excellent performance when classifying unstructured data such as images, streaming data, or for object detection. However, many studies argue about their performance in classifying tabular data. For instance, Intel advocates [17] that XGBoost shows better classification performance than DNNs when dealing with tabular data. This is confirmed by [9], where authors justify the supremacy of tree-based classifiers with respect to deep learners when processing tabular data stating that they adapt well to *specific features of tabular data: irregular patterns in the target function, uninformative features, and non-rotationally-invariant data where linear combinations of features misrepresent the information.* Conversely, authors of [11] present a DNN which is optimized for tabular data and outperforms tree-based classifiers in some datasets. Similarly, authors of NODE [12] claim that their method is the first successful example of DNN that substantially outperforms gradient-boosting classifiers on tabular data. FastAI [13] can efficiently classify tabular data thanks to a custom pre-processing of features, which are treated differently whenever they describe categories or continuous numerical values.

There are even studies that aim at converting tabular data into images to transfer the potential of DNN in processing images to tabular data [18,19]. One of the most promising approaches is DeepInsight, which "converts non-image samples into a well-organized image form. Thereby, the power of DNNs, including GPU utilization, can be realized for non-image samples". DeepInsight enables feature extraction through "the application of convolutions for non-image samples to seize imperative information and shown promising results" [19]. This approach is radically different from any DNN for tabular data that was invented to date, as it does not act on the classifier, but on the input data. As shown in Fig. 1, DeepInsight can transform any tabular data into a pixel matrix through a feature re-ordering according to correlation, subspace mapping (using either t-SNE or PCA [42]), basic sub-setting, shifting and rotation image manipulations, and finalization of the pixel map. Noticeably, this process does not depend at all on the (image) to be used at a later stage, which can be any image classifier, either customized or obtained by transfer-learning from available models. Also, this methodology enables the classification of tabular data by using DNNs that are meant to classify images, thus exploiting their full convolution potential.

2.4. Diversity and machine learning

A common way of enhancing classification relies on ensemble learning: creating ensembles of individual classifiers for building a unified meta-classifier [43] or pipeline [44] often improves classification performance at a cost of increased processing time and model complexity. On the other hand, adopting ensembles of similar classifiers may not improve classification accuracy [39], because they will misclassify the same data points in the same way. Intuitively, if an error or intrusion is not detected by any of the classifiers in an ensemble, it is not possible to detect it by just ensembling independent classifiers, no matter how we combine their results. Using terminology specific to error and intrusion detectors, we refer to this event as a common mode failure [7], where all components tasked with detection agree on a prediction that is wrong. a

That is why studies as [45–47] define and establish different ways to quantitatively estimate how a given pair of classifiers is similar or diverse based on their scores on one or more datasets. Moreover, the study [48] summarizes three different principles that drive the diversity of classifiers: Data, Model, and Inference Diversification. Data diversification provides different training data to many instances of the same classifier, or single instances of many classifiers. This concept is widely applied in Bagging (e.g., Random Forests) or Boosting (e.g., XGBoost) classifiers, and makes them more accurate than their Decision Tree baseline [49,50]. Model diversification can be implemented either by creating different instances of the same classifiers that are trained using different parameters (e.g., a different k value for kNN) or by using ensembles of different classifiers. Lastly, inference diversification is concerned with obtaining multiple outputs in the inference of the classifier. Practically speaking, the classifier should output a set of possible outputs alongside a ranking or confidence scores. Whereas this last diversification approach has been proven useful for image classification, it does not apply to the analysis of tabular data and especially to error and intrusion detectors, which require a single unambiguous prediction. On the other hand, two different classifiers may behave similarly on specific input data, with detrimental effects on the discriminative power of the ensemble.

Once a diverse set of classifiers gets defined, there needs to be a way to combine them to obtain a unified prediction for the ensemble of

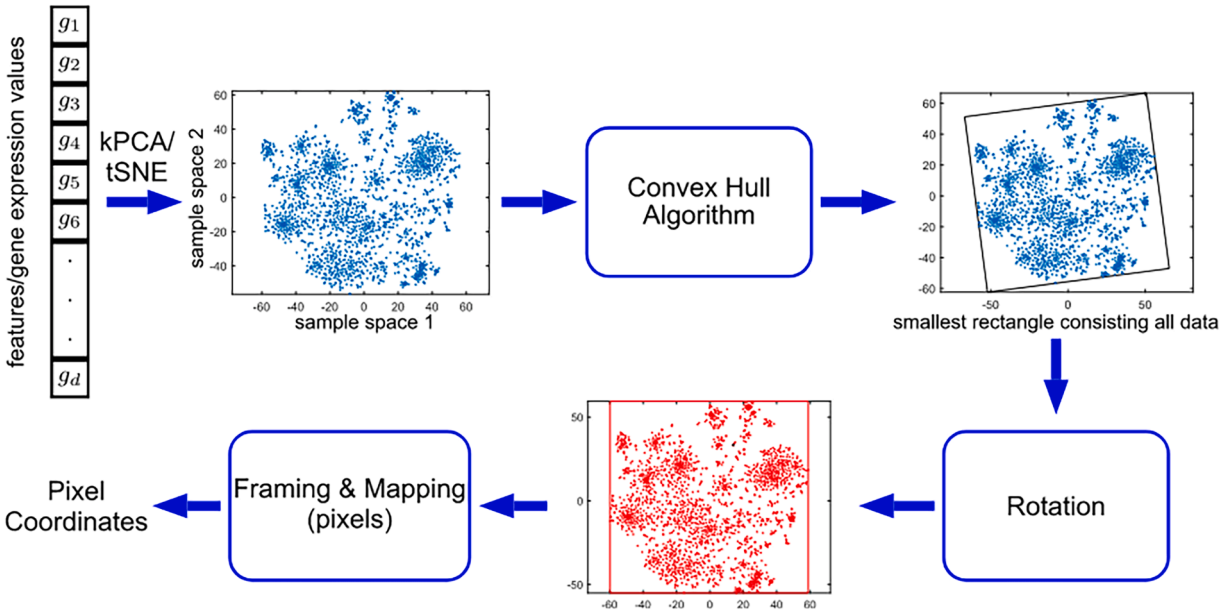


Fig. 1. DeepInsight methodology for converting tabular data to images [19].

classifiers. A meta-learner is a complex classifier that uses knowledge acquired during base-learning episodes, to improve meta-level classification capabilities [43]. Its base-learning process feeds dataset features into one or more classifiers that will output model-based meta-features [51] to be provided alongside other features to the meta-classifier, which computes a unified result. Different ways to combine ensembles of classifiers such as cascading, stacking, (weighted) voting, delegating, and arbitrating have been proposed through the years and recently summarized in [21]. Majority voting is used to combine results obtained by bagging and boosting ensembles. Stacking has been proven to be effective for combining ensembles of heterogeneous classifiers [52,53], whereas cascading, delegating, and arbitrating are not easy to orchestrate and as a result have been used sparingly. In [54] authors conduct experiments and conclude that stacking outperforms voting for model combination, and has comparable classification performance with bagging. Conversely, combining ensembles of classifiers to obtain a unified class prediction is not trivial and does not always result in improved capabilities as some misleading classifiers may let the meta-learner lean towards a misclassification [52].

2.5. Machine learning for (Safety) critical systems

The development of (safety-)critical systems relies on stringent safety methodologies, designs, verification, and validation activities to prevent the occurrence of catastrophic failures. These activities are regulated by various standards: ISO26262 and ISO/PAS 21,448 (SOTIF – Safety Of The Intended Functionality) for automotive, CENELEC EN5012x for railways, DO178x for avionics, and the general IEC61508 for generic hardware-software systems. Specifically for software development, the process ensures traceability across requirements, architectural and unit design, code, and verification. System-wise, the likelihood of critical failures should not exceed a specific threshold, usually a Safety Integrity Level (SIL). This is a system-level property, but it is tightly coupled with the behavior and the interconnections of individual components [7]. Thus, sub-systems or components – including classifiers - should be proven to have a rate of critical errors or failures below a specific threshold. Particularly, the SOTIF standard recognizes performance limitations of software (including and especially for ML-based components) and expects that the scenarios/inputs that belong to unsafe-unknown (e.g., samples out of training distribution) and unsafe-known (e.g., samples out of operational design domain) situations shall be reduced to the extent that the frequency of misclassifications due to these events is considered acceptable [55]. This is very important as the robustness of classifiers to unexpected, or non-Independent and Identically Distributed (non-IID) data, is usually very challenging to achieve [10,56,57].

Importantly, critical systems are first designed with safety, security or reliability in mind. This means that the system has to be conceptualized and designed in a way that does not expose users to critical hazards or threats. Once these properties are guaranteed, system architects and engineers start optimizing desirable properties as availability and quality of service. This has an important cascading effect on the way classifiers can be integrated into a critical system: we don't want classifiers that are always correct, but we favor those that have the least amount of misclassifications, especially those that have direct consequences for the encompassing system. Having a high percentage of correct classifications is clearly desirable, but it is not a must-have condition for bringing classifiers into critical systems.

3. Motivation and novelty of the paper

In real-world applications and especially critical systems, the most common data type is tabular data, comprising samples (rows) with the same set of features (columns) monitored from a system. Thus, recent years have seen a growing interest of critical system architects and engineers in exploring classifiers for solving complex problems including, but not limited to, anomaly-based error and intrusion detection. Deploying a classifier for a critical system is already very challenging due to compliance with standards. Moreover, designing, training, and testing classifiers that suit a given functional and non-functional requirements is always a problem-specific task that requires many experiments, sensitivity analyses, and fine-tuning which cannot be avoided.

This paper aims to minimize the effort in devising the optimal tabular data classifier for a specific system and task. Although the no free lunch theorem ("there is no algorithm that can solve all optimization problems better than others" [58]) universally holds, we aim at identifying a subset of classifiers that regularly (but not always) outperform others for anomaly-based error and intrusion detection, either for classification performance, faster training times or lesser resource (e.g., storage) consumption. Our comparison includes individual classifiers and ensembles that use diverse base-learners, and adequate meta-learning strategies.

This study has obvious practical implications: reducing the set of candidate classifiers for building detectors is going to speed up experimental analyses, allowing an early completion of the process, or freeing up time that could be used to fill documentation for standards compliance. To accomplish that, we gather many tabular data classifiers, and exercise them alone and in conjunction, trying to reach clear and conclusive statements that build upon recent conjectures from various papers, most notably [9,17].

4. Comparison of tabular data classifiers

This section compares individual classifiers on different tabular datasets that relate to binary and multi-class classification problems in critical systems. To accomplish that, we gather and preprocess 23 public datasets containing data monitored from ICT systems that report normal data points alongside data collected during the occurrence of errors, attacks, or failures (see Section 4.1). Then, we prepare tree-based, Tabular DNN and Image classifiers that will be exercised on each of the datasets above using a 50–10–40 train-validation-test split. Details on the classifiers and hyper-parameters are in Section 4.2. Lastly, we will collect metrics about the classification performance in Section 4.3.

Experiments have been executed on a Dell Precision 5820 Tower with an Intel Xeon Gold 6250, GPU NVIDIA Quadro RTX6000 with 24GB VRAM, 192GB RAM and Ubuntu 18.04, NVIDIA driver 45.119.03 with CUDA 11.

4.1. Error, attack and failure datasets

There are a wide variety of data to be classified to improve ICT systems, ranging from devices data in Internet-of-Things (IoT) or Industrial Control Systems (ICS), network data for intrusion detection, or hardware monitoring data. Amongst those many alternatives, we consider 23 datasets as data baseline for this study: 11 datasets of network intrusion detection, 5 datasets related to hardware monitoring for failure prediction, and 6 datasets related to error and anomaly detection in IoT and ICS systems. Table 1 summarizes the datasets involved in this study, reporting domain, name, year, number of data points, number of features, and categories of anomalies, errors or attacks. All datasets are labelled, in CSV format, and were cropped to 200 000 items for the feasibility of our study, which would have taken months of server execution otherwise. However, we exercise classifiers in a big portion of two datasets for big data performance analyses in Section 4.5.

4.1.1. Network intrusion detection (NIDS)

We selected labelled datasets on network intrusions looking in surveys [59], *Kaggle, UCI, Zenodo, IEEEDataport* and other online portals. Our selection process resulted in the following datasets: ADFANet [60], AndMal17 [61], BAIoT Doorbell [62], CICIDS17 [63], CICIDS18 [63],

Table 1
Name, release year, number of attack types, number of portions, and the amount of ordinal features *f* of used datasets.

Domain	Dataset Name	Year	Categories of Anomalies	# Features	Number of Data Points
Network Intrusion Detection	ADFANet	2015	5	3	132 002
	AndMal17	2017	4	75	100 522
	BAIoT Doorbell	2018	5	115	75 165
	CICIDS17	2017	4	75	200 000
	CICIDS18	2018	5	75	200 000
	CIDDS	2015	4	7	200 000
	IoT Network	2019	9	8	210 425
	ISCX12	2013	4	6	200 000
	NSLKDD	2009	4	37	148 517
	UGR16	2016	5	7	207 256
	UNSW-NB15	2015	8	38	165 461
HW Monitor	BackBlaze 2017	2017	1	50	32 678
	BackBlaze 2019	2019	1	44	47 525
	BackBlaze 2021	2021	1	37	44 950
	BackBlaze 2023	2023	1	35	70 512
	BAIDU	2017	1	12	186 049
Error / Anom. Detection	Arancino Device	2023	9	119	154 000
	HAI Pressure	2019	1	54	200 000
	HAI ICS	2023	1	224	54 000
	MAFAULDA	2018	1	8	200 000
	Mechanical Failure	2018	1	18	7 906
	Metro PT	2022	2	20	173 824
	Scania Trucks	2016	1	170	76 000

CIDDS [64], IoT Network [65], ISCX12 [66], NSLKDD [67], UGR16 [68], UNSW-NB15 [69]. All those datasets report normal data points and data points collected while the system is under attack. Features are mostly numeric features extracted by monitoring network flows and packets (e.g., bytes received per second, number of packets).

4.1.2. Hardware failure prediction

Classifiers may also spot anomalies that could potentially anticipate the failure of hardware components. To include that, we gathered datasets related to performance monitoring of hard disks that label each data point as corresponding to failure if the monitored hard drive was in a fail state or going to fail thereafter. BackBlaze [70] makes many years of hard drive data available to the public, reporting labeled data related to many SMART indicators of hard drives, while another source of hard drive data came from the BAIDU [71] competition whose input datasets are still available for download.

4.1.3. Error/Anomaly detection

The last group of datasets we consider comes from IoT or ICS systems: a distributed control systems of a power plant controlling a turbine [72,73], malfunctions of metros in Portugal [74], railroad trucks equipped for sensors to monitor brake pressure [75], an edge device monitored for errors [76], the mechanical failure of electrical machinery in power plants [77], and a simulated multivariate time-series acquired by sensors on a SpectraQuest's Machinery Fault Simulator [78].

4.1.4. Preprocessing

We transform the tabular datasets into CSV files with a tabular structure. ISCX12, IoT Network, and UNSWNB15 are available only as a collection of monitored PCAP network packets, which we convert in CSV format using *tshark*. Then, we remove features that are specific of the setup that was followed to gather data, namely: Timestamp, ID, experiment number, if any. Those features should be disregarded for classification purposes as they carry information about the experiments to build the dataset: classifiers using these features may learn how experiments were made instead of how the system behaves.

The BackBlaze manufacturer provides data in CSV files aggregating three months of log data in each file. Each of these CSVs contains extremely unbalanced data with just a few failure logs (<0.01 %). To create the datasets for a specific year, we merged 4 CSV files referring to all the months of 2017, 2019, 2021, 2023 years and we down-sampled the normal observations to reach a balance of 98 % normal – 2 % failed data. Then, we zero-filled all blank values. This allows for mitigating the data imbalance problem and provide solid data for classification. The error/anomaly detection datasets mostly come structured as single CSV files that are ready to be used for classification. The only exception being the MetroPT dataset, which contains separate files for different monitoring periods. To obtain a single dataset containing normal data plus OilLeak and AirLeak failures we concatenate 3 different files containing i) only normal data, ii) normal data plus OilLeak data, iii) normal data plus AirLeak.

4.2. Classifiers and hyper-parameters

Our experimental campaign compares classifiers belonging to three groups: traditional (non-DNN) supervised classifiers, tabular DNN classifiers, and image DNN classifiers that we apply after transforming tabular data points into images.

4.2.1. Traditional supervised classifiers

According to the results in [20], we selected non-DNN supervised classifiers that are known to have good classification performance. We ended up selecting eight classifiers, briefly described in the following. For each classifier, we include an acronym, that we will use in the rest of the paper.

- A Decision Tree (DT, [79]) learns simple decision rules inferred from the data features, by building a tree that can be seen as a piecewise constant approximation.
- Random Forests (RF, [35]) is a combination of Decision Trees through Bagging, which independently samples random vectors with the same distribution for all trees in the forest, and aggregates results by voting or averaging.
- XGBoost (XGB, [80]) is an optimized gradient boosting method, i.e., gives a prediction model in the form of an ensemble of weak learners called decision stumps.
- ExtraTrees (ET, [36]) is an ensemble of randomized decision trees (a.k.a. extra-trees) that are fit on various sub-samples of the dataset and – similarly to Random Forests - use averaging to improve the predictive accuracy and control over-fitting.
- LogitBoost (LB, [81]) is a boosting algorithm based on additive logistic regression, where the objective is to minimize the logistic loss trough subsequent training of weak logistic regression models.
- Naïve Bayes (GNB, [82]) exploits the Bayes' theorem under the naïve assumption that the features are conditionally independent, given the target class, to build a classifier.
- Logistic Regression (LR, [82]) is a statistical supervised classifier that models the logit of an event as a linear combination of one or more independent variables.
- Linear Discriminant Analysis (LDA, [83]) assumes that all classes are linearly separable. Then, multiple linear discrimination functions (i.e., Fisher's Discriminants) representing hyperplanes in the feature space are created to distinguish the classes.

Most of these classifiers can be run using *scikit-learn, xgboost, logitboost* libraries with default parameters and minimal customization, avoiding potential misconfigurations by the user. Thus, we will not be performing sensitivity analyses to choose the optimal value of

Table 2
MCC Scores for each classifier in each of the datasets. Average MCC and training time (in seconds) at the bottom.

		Traditional Supervised (non-DNN)								Tabular DNN						Image DNN			
Domain	Dataset Name	GNB	LDA	LR	DT	RF	XGB	ET	LB	FAI	TN	NODE	Tnet	GATE	CE	DI_ PCA	DI_TSNE	DI_MN	DI_MNIST
Network Intrusion Detection	ADFANet	.477	.124	.695	.903	.904	.905	.903	.904	.070	.050	.437	.250	.310	.484	.875	.864	.458	.105
	AndMal17	.014	.049	.022	.557	.511	.459	.543	.203	.176	.000	.000	.000	.199	.153	.143	.142	.000	.000
	BAIoT Doorbell	.855	.998	.000	1.0	1.0	1.0	1.0	1.0	.998	.926	.880	.636	.935	.937	.999	.999	.998	.483
	CICIDS17	.618	.849	.667	.995	.996	.998	.990	.983	.947	.922	.940	.764	.910	.939	.958	.956	.950	.381
	CICIDS18	.760	.852	.736	.889	.899	.905	.895	.906	.896	.881	.891	.459	.892	.896	.896	.896	.898	.510
	CIDDS	.187	.011	.337	.976	.976	.976	.975	.968	.468	.482	.664	.465	.684	.443	.799	.812	.482	.017
	IoT Network	.128	.501	.286	.959	.959	.960	.959	.964	.871	.803	.682	.534	.766	.851	.830	.871	.866	.098
	ISCX12	.120	.004	.025	.861	.868	.867	.867	.753	.517	.026	.348	.003	.558	.580	.560	.566	.397	.000
	NSLKDD	.280	.872	.633	.992	.996	.996	.995	.986	.991	.979	.941	.909	.971	.970	.984	.984	.979	.664
	UGR16	.043	.000	.004	.879	.884	.883	.883	.790	.000	.000	.000	.000	.000	.000	.000	.000	.000	.000
	UNSW-NB15	.253	.427	.076	.706	.729	.717	.733	.615	.648	.565	.545	.413	.470	.598	.587	.621	.588	.000
HW Monitor	BackBlaze 2017	.499	.439	.000	.481	.642	.641	.642	.629	.624	.470	.618	.527	.596	.459	.514	.576	.573	.000
	BackBlaze 2019	.470	.436	.000	.506	.662	.669	.636	.659	.587	.479	.578	.272	.580	.445	.554	.631	.558	.000
	BackBlaze 2021	.442	.379	.000	.718	.773	.765	.793	.660	.676	.449	.525	.227	.382	.555	.660	.629	.604	.000
	BackBlaze 2023	.299	.375	.000	.692	.765	.784	.770	.704	.607	.575	.546	.354	.010	.547	.654	.631	.604	.000
	BAIDU	.791	.792	.834	.995	.995	.996	.996	.983	.990	.968	.963	.948	.973	.982	.990	.988	.981	.566
Error / Anom. Detection	Arancino Device	.116	.579	.344	.686	.774	.793	.808	.664	.720	.641	.626	.469	.683	.640	.660	.593	.579	.123
	HAI Pressure	.749	.785	.788	.964	.984	.983	.992	.888	.985	.940	.872	.818	.971	.973	.964	.971	.955	.000
	HAI ICS	.614	.617	.630	.946	.958	.952	.979	.796	.961	.848	.838	.587	.947	.938	.945	.943	.842	.000
	MAFAULDA	.657	.000	.000	.793	.867	.884	.874	.785	.884	.859	.713	.000	.841	.723	.866	.864	.798	.158
	Mechanical Failure	.765	.798	.780	.606	.849	.849	.470	.881	.816	.000	.780	.000	.000	.472	.745	.798	.248	.000
	Metro PT	.755	.795	.814	.981	.992	.994	.993	.897	.906	.823	.785	.725	.823	.870	.846	.896	.845	.218
	Scania Trucks	.538	.723	.643	.706	.788	.807	.780	.762	.718	.383	.539	.026	.706	.438	.731	.670	.614	.000
Average MCC		*.454*	*.496*	*.362*	*.817*	*.861*	*.860*	*.847*	*.799*	*.698*	*.568*	*.640*	*.408*	*.618*	*.648*	*.729*	*.735*	*.644*	*.144*
Average Train Time (s)		*< 0.1*	*< 1*	*116*	*1*	*7*	*10*	*8*	*77*	*51*	*64*	*32*	*54*	*1882*	*31*	*108*	*133*	*967*	*276*

parameters of these classifiers, and just use them with their default parameters. An exception is represented by GNB, which requires scaling the value of dataset features for optimizing the decision boundaries. Also, LR delivered many warnings about a failed convergence using its default setup. Thus, we ran grid searches with *HyperOpt* [84], which uses Bayesian optimization and has pre-built interfaces for many classifiers in *scikit-learn*. Even in this case, we could not avoid this problem of failed convergence in all cases; however, the combination of parameters that minimized them was using *solver* = *'sag'*, max_*iter* = *1000* and *tol* = *0.001*.

4.2.2. DNN for tabular data

After surveying recent works and frameworks, we selected a set of 6 DNNs for tabular data using six different frameworks.

- *FastAI* (FAI [13]), used through the implementation available for tabular data *autogluon-tabular*, available at https://auto.gluon.ai/stable/api/autogluon.tabular.models.html#autogluon.tabular.models.NNFastAiTabularModel). FastAI is a deep learning library that automates the creation of models; it includes a set of optimizations that are automatically selected to apply DNN on tabular data.
- *TabNet* [11] (Tnet, *pytorch-tabnet* implementation at https://dreamquark-ai.github.io/tabnet/) is a DNN for tabular data which, amongst the innovative aspects, uses sequential attention to choose which features to reason from at each decision step.
- *NODE* [12] (*pytorch-tabular* implementation https://pytorch-tabular.readthedocs.io/en/latest/models/) is a DNN which exploit ensembles of oblivious decision trees, in a CatBoost-like fashion.
- *GATE* [85] (*pytorch-tabular* implementation) exploits a gating mechanism as a feature representation learning unit with an in-built feature selection mechanism. The authors combine it with an ensemble of differentiable, non-linear decision trees, re-weighted with simple self-attention to predict the desired output.
- *Category Embedding* (CE, *pytorch-tabular* implementation) is a feed-forward network where the categorical features are embedded through a learnable embedding layer.
- *TabNet from Pytorch-tabular*: the library *Pytorch-tabular* includes another implementation of TabNet, which we include as well (TN).

The DNNs above use the following hyperparameters:

- FastAI performs an internal hyper-parameter optimization process, which is connected to the pre-processing and feature learning stages in the early steps of the training process.
- For Tnet, we ran grid searches with 54 combinations of the following parameters and values:
  - Learning rate [$e-5$, $e-3$, $e-1$],
  - Batch size [128, 256, 512]; bigger batch sizes were likely to create GPU memory issues,
  - Max Epochs [20,50, 100],
  - patience (for early stopping) [2,5].

The *pytorch-tabular* framework does not allow setting patience, thus we discarded that option for TN, performing grid searches only on other parameters (total of 27 combinations).

- For NODE and GATE we used the values of parameters suggested in the respective studies [12,85], as authors explain that they provide a good tradeoff between classification speed and classification performance.
- CE was set to use the following parameters:
  - Learning rate [$e-5$, $e-3$, $e-1$],
  - Layers [1024–512–256, 512–256–128]

For each classifier and each dataset, we selected the configuration that ended up having the highest MCC (see Section 4.3) on the validation set.

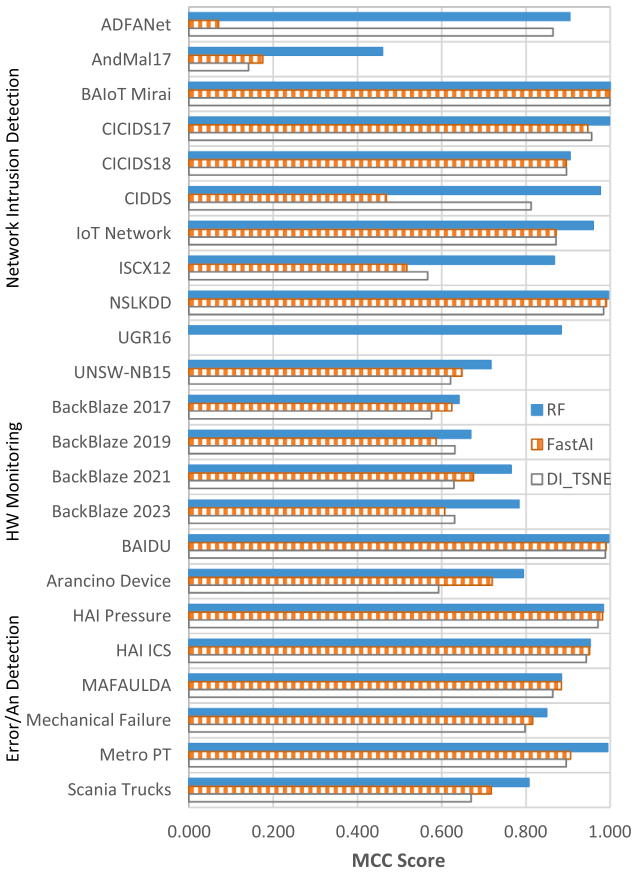


Fig. 2. MCC scores of Supervised (blue bar), tabular DNN (orange-sliced bar) and image DNN classifiers (empty bar with gray border) on each dataset in the study.

4.2.3. DNN for image classification

We first explain the setup of DeepInsight. As already motivated in Section 2.2, tabular datasets related to errors, intrusions, and failures of systems do not usually have thousands of features. Thus, it does not make sense to transform tabular data into big images. Ideally, we want our informative content to be readily available to classifiers instead of being scattered far away in big images. Thus, we set DeepInsight to convert tabular data into images of $32 \times 32 \times 3$ size, using either t-SNE or PCA in the process. Then, we design a custom DNN composed of a conv2d $32 \times 32 \times 3$ layer, output, a batch normalization, maxpooling2d, flattening step, a dense layer of size 32, a dense layer of size 16, and then a final layer that has the same number of neurons as the number of classes of the problem. To limit overfitting, we employ a 20 % dropout between dense layers. These classifiers are called DI_TSNE and DI_PCA, depending on whether the tabular-to-image process uses t-SNE (DI_TSNE) or PCA (DI_PCA).

Then, we are interested in having classifiers that are not trained from scratch, but that transfer learn from existing models trained on common benchmark datasets. Out of the many available models, we opt for small DNNs as MobileNetV3 since, again, the information density in images obtained by DeepInsight may be scarce and thus using a complex DNN may i) be classification overkill, and ii) result in a training process that overfits on the training set, having very good train loss but poor classification performance on a different test set. We use *imagenet* weights from Keras with the smallest image size (128) available, which is also the target image size we set DeepInsight to work on in this specific case.

Last, we create a classifier that transfer learns from a DNN from MNIST, which is an old and well-known benchmark dataset of 28×28

images. The resulting classifier DI_MNIST will load weights from a model trained on MNIST [86], and top it with a single dense layer with as many neurons as the number of classes in the problem (which are always less than 10 in our datasets). This last classifier is the only one that processes 2D-pixel maps and uses a simpler model than DI_MN: it will not likely overfit during re-training, but it may not be complex enough to catch all the nuances of a tabular classification task.

4.3. Evaluation metrics

Classification performance is usually measured through the confusion matrix and compound metrics such as accuracy, i.e., the fraction of correct predictions over all predictions. However, accuracy (ACC) may deliver misleading results when datasets are unbalanced [87,88], which happens frequently in error and intrusion detection: therefore, we will mostly rely upon the Matthews Coefficient (MCC), which does not suffer of the above problem and has a flexible formulation that scales well to both binary classification and multi-class classification problems.

4.4. Results and discussion

Exercising the experimental methodology led to a massive amount of results about the classification performance of different classifiers. We examine these results using Table 2 and Fig. 2.

Table 2 shows MCC scores achieved by each traditional (non-DNN) supervised, tabular DNN, and image DNN classifier on each of the 23 datasets considered in this study. The two rows on the bottom of the table report the average of these MCCs over datasets, and the average training time (in seconds) required to train these models. Traditional tree-base classifiers DT, RF, ET, XGB and LB show the best classification performance with an average MCC over 0.8 and typically require only a few seconds to train (e.g., an average of 10 s is needed to train an XGB classifier). Classifiers as LR. GNB and LDA show very fast training times (often below one second), but they output more misclassifications than their competitors, thus they have a sub-optimal MCC. MCCs of tabular DNNs appear in the middle of Table 2, with FastAI having the highest MCC of the group. GATE is the classifier that takes the most time to train (1882 s), with an MCC that is inferior to those of other tabular DNNs and, consequently, of tree-based supervised classifiers.

Image classifiers applied after tabular-to-image conversion take more time to train compared to tabular DNN classifiers. DI_PCA and DI_TSNE, the image classifiers trained from scratch, have higher MCC than any tabular DNN. Image classifiers obtained using transfer learning from the MobileNet model (DI_MN) perform similarly to tabular DNNs ad CE or NODE, but take far more time for training (average of 967 s for DI_MN against 31 and 32 for CE and NODE, respectively). The DI_MNIST classifier shows poor classification performance as the transfer learning process often results in learning a model that always predicts the same class, thus obtaining many MCC = 0 (see last column of Table 2). This is probably due to the trivial complexity of a network trained on MNIST and having only an additional dense layer on top.

Generally speaking, whenever a classifier shows an MCC = 0 in Table 2 it means that the training process did not converge to a model that is capable of classification. This happens quite frequently with DI_MNIST (as discussed above), but also occurs for LR in all BackBlaze datasets due to the extremely low amount of “anomalous” data point instances (approximately 2 %), which is a well-known weakness of this classifier [89]. Poor classification performance can be observed for the UGR16 dataset, where all DNNs have an MCC of 0. Our explanation is as follows: while training tabular and image DNNs, the validation loss is gradually converging to the final value, but has an erratic behavior instead. Changing the loss function allows for not having an MCC = 0 in this dataset, but worsens the classification performance in all other datasets. Thus, we decided to keep results as in Table 2.

Another viewpoint on these experimental results is provided by Fig. 2, which is a bar chart that reports, for each dataset, MCC scores of RF, FastAI, DI_TSNE, which have the highest average MCCs of 0.861, 0.698, 0.735 in their respective categories (traditional supervised, tabular DNN, image DNN). While average scores in Fig. 2 provide a first view of classification performance, it is beneficial to look at the scores on each dataset to get insights and explain specific trends. Particularly we can observe that the blue solid line of RF is always the longest, or it always shows the best MCC (often on par) on each dataset. The only counter-example is the MAFAULDA dataset, where FastAI slightly outperforms RF. However, even if not shown in the bar chart for brevity, XGB outperforms both RF and FastAI in MAFAULDA. The direct consequence of this observation is highlighted below.

Takeover 1. DNNs never outperform tree-based supervised classifiers in intrusion/error/anomaly detection datasets used in this study.

Moreover, it turns out evident that there are datasets in which the superiority of RF against FastAI and DI_TSNE is astounding: it is the case of AndMal17, ISCX12, UGR16 and, to a lesser extent, ADFANet and CIDDS (especially comparing against FastAI). All these datasets report on packet or network flow data for intrusion detection, and have a normal class plus many labels corresponding to attacks: 4 for CIDDS, AndMal17 and ISCX12, 5 for ADFANet and UGR16. Also, they have a varying number of features that ranges from 3 (ADFANet) to 75 (AndMal17). However, other datasets such as CICIDS18 have labels corresponding to 5 attacks and 75 features – thus comparable with those above – but the classification performance of RF, FastAI, and DI_TSNE is very similar. Thus, the difference in classification performance between these three classifiers is not solely due to the number of features nor to the number of classes of the problem. Also, the difference in the UGR16 dataset is due to the training phase of DNNs not converging at all, resulting in a model that is always predicting the normal class for any data point, thus MCC = 0, or random guessing for a 6-class problem. When this happens, it is usually a matter of DNN structure or values of the hyperparameters, including loss function, epochs, batch size, stop conditions, or even pre-processing of data (e.g., normalizing). However, the structure of tabular DNNs is either automatically devised by frameworks or coded to be optimal for classifying tabular data, and the grid searches we ran using different values for hyperparameters did not lead to better performance. Thus, the failed convergence of DNNs could be due to many different problems that are not easy to address and lead to our Takeover 2.

Takeover 2. DNNs have many hyperparameters that should be assigned after fine-tuning through tailored and algorithm-specific sensitivity analyses. These may be very time-consuming and require domain-specific knowledge that may be not available or costly.

This is another reason why tree-based classifiers, which require minimal parameter tuning (e.g., number of trees in a RF, max depth of trees) and build models that are easier to understand [90], should be the preferred choice when classifying tabular data.

4.5. Is it a matter of data size?

These results may raise legitimate questions about the impact of the size of the training set used in experiments. The expectation is that DNNs would perform better when more features or more training data are available since they are complex models with potentially millions of weights. Providing more data is not always the solution and may result in overfitting; on the other side results in the previous section may be subject to underfitting.

Thus, we re-ran the classifiers that had better classification performance from the previous section using subsets of the ISCX12 and MAFAULDA datasets containing a growing amount of data points, having a train set size of {1 000, 2 000, 5 000, 10 000, 20 000, 50 000, 100 000, 200 000, 500 000, 1 000 000}. We chose these two datasets as they belong to two different domains and are big enough in their raw formulation to extract up to almost two million data points (train,

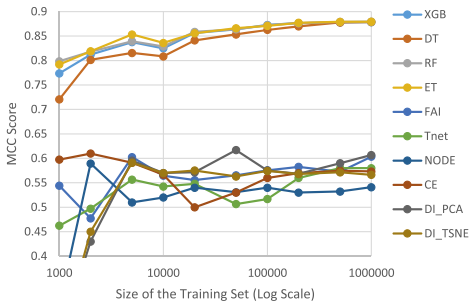


Figure 3a: MCC in ISCX12

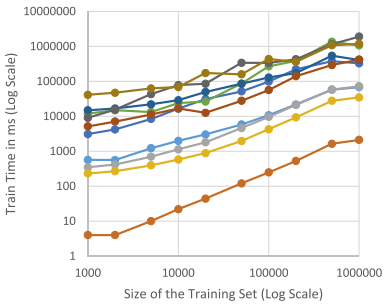


Figure 3b: Train Time in ISCX12

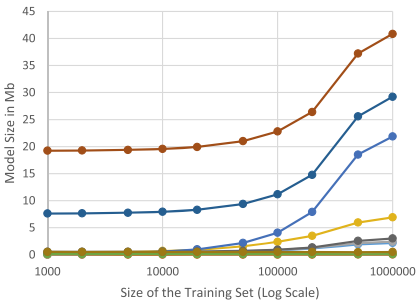


Figure 3c: Model size in ISCX12

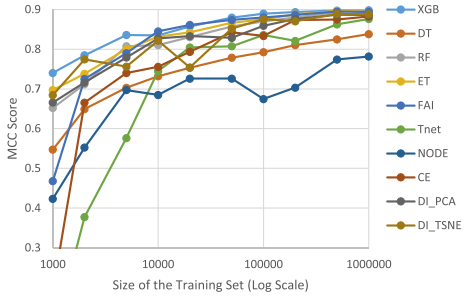


Figure 3d: MCC in MAFAULDA

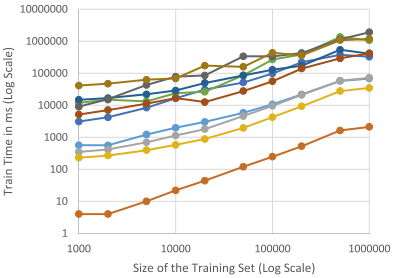


Figure 3e: Train Time in MAFAULDA

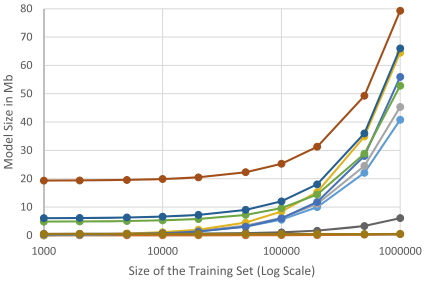


Figure 3f: Model size in MAFAULDA

Fig. 3. Evolution of MCC scores (left), train time (center), model size (right) for varying train sizes. On top of the figure (4a, 4b, 4c) are results for ISCX12, bottom (4d, 4e, 4f) are results for MAFAULDA.

validation, and test). Also, while for ISCX12 there is a clear difference between the performance of tree-based classifier and DNNs, for MAFAULDA the classification performance is almost on par.

The results of this study are in Fig. 3, which reports 6 plots. From left to right of the figure, we observe the evolution of MCC scores, train time (ms) and model size (MB) while using training sets of growing size for ISCX12 (up in the figure, Fig. 3a to Fig. 3c) and MAFAULDA (Fig. 3d to Fig. 3f). Compared to results on the 100 000 x-axis mark, which were presented in the previous section, we can observe quite common trends for all classifiers. All of them generally had a higher MCC when trained using big train datasets (aside from small fluctuations), meaning that all classifiers still had room to learn without overfitting on the training set (see Fig. 3a, Fig. 3d). Trends observed while using a training set of 100 000 items hold also using bigger train sets: there is no competitor that was behind others using smaller train sets and jumps ahead when using big datasets.

Takeover 3. DNNs do not outperform tree-based ensembles even when processing "big data" tabular training datasets. While the size of the training set matters for the overall classification performance, using training sets containing more or less items does not change the relative ordering of classifiers for a specific problem (using MCC as reference)

The train time in Fig. 3b and Fig. 3e, and the model size in Fig. 3c and Fig. 3f grow homogeneously, without showing erratic trends nor anomalous spikes. Whereas the size of the model learned at the end of the training phase is very classifier-dependent, it is easy to observe how the training time required by DNNs (both tabular and image) is at least an order of magnitude more compared to tree-based classifiers, even using a server with adequate computing resources.

Takeover 4. DNNs take at least an order of magnitude (10x) more time for training than tree-based ensembles. Tabular DNNs are on average faster than image DNNs trained on images converted from tabular data points using DeepInsight.

5. Classifier ensembles and meta-learning

Whereas it is pretty clear that tree-based classifiers outperform DNN classifiers for tabular data, we want to take our analysis one step further. Instead of considering DNNs and tree-based classifiers as challengers, we conjecture that they may team up particularly well and build anomaly-based error and intrusion detection with potentially lower

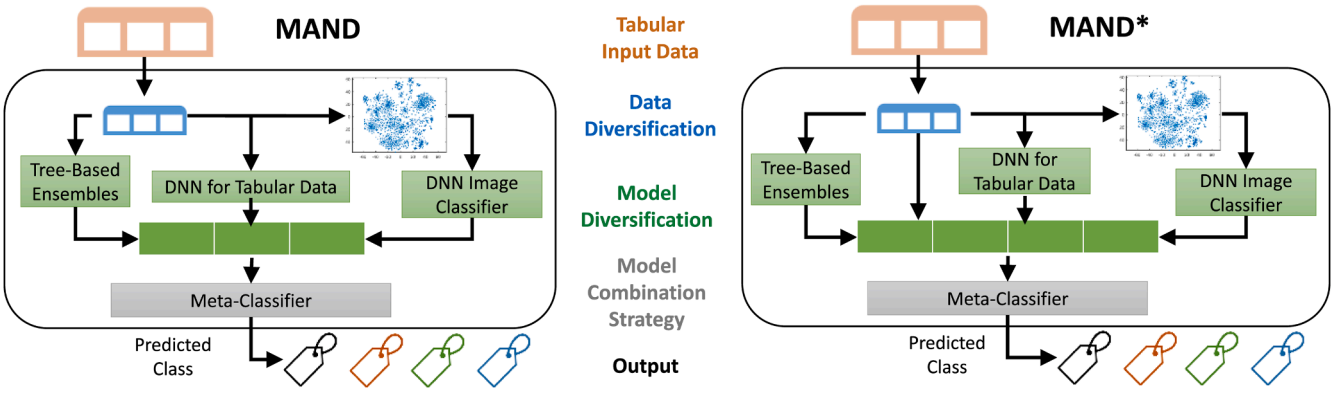


Fig. 4. MAND (left) and MAND* (right) classifiers for combining different classifiers and enhance error and intrusion detection.

misclassifications than considering them individually. To test our conjecture, we craft ensembles that include a tree-based classifier and a tabular DNN as base-learners. The classifiers above guarantee model diversification [48] of our ensemble: trees and DNNs are conceptually diverse since they i) build different models, ii) only DNNs learn complex features during training, and iii) decision trees are often used to build ensemble classifiers, whereas DNNs contain a single strong model for classification. On the negative side, providing the same data to trees and tabular DNNs does not guarantee strong data diversification, albeit tree-based ensembles perform bagging or boosting processes that implement a weak type of data diversification. As such, we foresee the usage of a third base-learner: an image DNN that performs image classification which we apply to tabular data after transforming data points to images.

Then, we combine the three base-learners using a meta-learning strategy. Obviously, this creates two levels of classification which are sequential by design: thus, the prediction time of the meta-classifier should be fast enough to avoid additional timing overheads to the error and intrusion detection tasks, which often operate in systems that deal with (hard) real-time settings [29,91]. We extract the following model-based features by processing the class probabilities provided by each classifier in the ensemble:

- probas: the array of probabilities
- maxp: the maximum probability assigned to any class;
- *ent*: the entropy of the array of probabilities;
- class: the class corresponding to the argmax of probabilities.

In a binary classification problem, the *maxp* and *ent* quantities convey the same information. However, this formulation applies to both binary and multi-class problems, for which the two quantities carry different information.

This conceptualization opens two possible design patterns for an ensemble classifier (see Fig. 4). The meta-classifier may use i) model-based features only, or ii) dataset features plus model-based features. The former approach builds a MAND (enseMble of trees ANd Dnns) classifier, while the latter ensemble classifiers are referred to as MAND*. Both classifiers are based on the Stacking architecture that was first presented by Wolpert in [22].

6. Ensemble assessment

The methodology for exercising MAND and MAND* classifiers uses the same datasets and the same classifiers in Section 4. This allows for comparisons and minimizes the risk for biasing experiments in one way or another.

6.1. Choice of the base-learners

MAND and MAND* classifiers require a set of three base-learners: a tree-based ensemble, a tabular DNN and an image DNN. Results from Section 4.5 can already suggest good candidates and classifiers that may be avoided as they have very poor classification performance. This is the case of LDA, LR, DI_MNIST, TN, whose average MCC in Fig. 2 is lower than 0.6. Moreover, we do not consider LB as a good candidate for supervised classifiers as it is similar to XGB but has lower classification performance. DT and GNB are left out as well since the first base-learners for MAND and MAND*ensembles has to be a tree-based ensemble. We also avoid the usage of NODE, which already constitutes a mixture of neural networks and decision trees and as such it may behave very similar to tree-based ensembles.

While this selection process guarantees the choice of different classifiers, there are no guarantees that they are diverse i.e., their output is really diverse for a relevant set of predictions. Thus, we compute the disagreement metric from [47], which authors define as “the ratio between the number of observations on which one classifier is correct and the other is incorrect to the total number of observations”. This can be easily obtained by using the results from Section 4, simply comparing predictions of couples of classifiers with respect to the label. Result in Table 3 show that the disagreement between classifiers from the same group is generally low: lower than 0.02 for tree ensembles (top left of the table), lower than 0.07 for tabular DNNs (center of the table) and lower than 0.04 for image DNNs (bottom right of the table). However, we can notice how FastAI and Tnet have very high disagreement compared to tree ensembles and, to a lesser extent, with image DNNs. The expectation is that MAND and MAND* ensembles using these two tabular DNNs have the best potential to outperform individual classifiers as they guarantee more behavioral diversity (i.e., disagreement) compared to, for example, GATE and CE.

6.2. Choice of meta-classifiers

As motivated in Section 5, the meta-classifier for MAND and MAND* ensembles should add a minimal overhead to the prediction process; thus, it has to be implemented as a rather simple function. Our options are the followings:

- A stacking [22,52] meta-level classifier: DT and statistical classifiers as GNB and LDA, which we excluded from base-learners, are fast and have good classification performance. We call these meta-classifiers *c:DT, c:GNB, c:LDA*.

Table 3
Disagreement between couples of classifiers to be used as base-learners for MAND and MAND* ensembles.

	dis	RF	ET	FastAI	Tnet	GATE	CE	DI_PCA	DI_TSNE	DI_IN
Tree Ens.	XGB	.017	.017	.077	.093	.058	.057	.035	.032	.055
	RF		.001	.091	.106	.072	.071	.049	.046	.069
	ET			.091	.106	.072	.071	.049	.047	.069
Tabular DNN	FastAI				.028	.071	.050	.066	.062	.049
	Tnet					.066	.059	.070	.076	.054
	GATE						.038	.035	.040	.035
	CE							.039	.037	.020
Img DNN	DI_PCA								.017	.037
	DI_TSNE									.033

Table 4
MCC scores of different MAND and MAND* classifiers varying base learners and meta-classifiers.

Base Classifiers	disagreement	MAND						MAND*		
		voting	avg	conf	c:DT	c:LDA	c:GNB	c:DT	c:LDA	c:GNB
RF - Tnet - DI_IN	0.115	0.683	0.707	0.744	0.860	0.861	0.768	0.861	0.861	0.740
ET - Tnet - DI_IN	0.115	0.683	0.695	0.750	0.838	0.793	0.789	0.838	0.802	0.778
ET - Tnet - DI_TSNE	0.114	0.735	0.743	0.771	0.845	0.829	0.789	0.841	0.831	0.781
RF - Tnet - DI_TSNE	0.114	0.745	0.752	0.765	0.862	0.861	0.770	0.863	0.861	0.750
RF - Tnet - DI_PCA	0.112	0.735	0.749	0.766	**0.864**	0.861	0.770	0.863	0.861	0.743
ET - Tnet - DI_PCA	0.112	0.728	0.736	0.772	0.832	0.822	0.790	0.840	0.826	0.782
RF - FastAI - DI_IN	0.104	0.740	0.769	0.786	0.860	0.861	0.761	0.859	0.861	0.741
ET - FastAI - DI_IN	0.104	0.730	0.769	0.792	0.841	0.837	0.783	0.842	0.838	0.773
ET - FastAI - DI_PCA	0.102	0.760	0.777	0.798	0.839	0.838	0.790	0.839	0.840	0.783
RF - FastAI - DI_PCA	0.102	0.763	0.777	0.792	**0.864**	0.861	0.763	**0.864**	0.861	0.745
XGBoost - Tnet - DI_IN	0.101	0.682	0.708	0.742	0.841	0.854	0.735	0.842	0.854	0.681
XGBoost - Tnet - DI_TSNE	0.101	0.744	0.757	0.762	0.842	0.854	0.737	0.844	0.853	0.698
ET - FastAI - DI_TSNE	0.099	0.758	0.778	0.798	0.841	0.839	0.792	0.835	0.841	0.785
RF - FastAI - DI_TSNE	0.099	0.759	0.776	0.793	0.863	0.861	0.770	0.863	0.861	0.751
XGBoost - Tnet - DI_PCA	0.099	0.733	0.751	0.760	0.845	0.854	0.734	0.845	0.854	0.689
XGBoost - FastAI - DI_IN	0.090	0.740	0.770	0.776	0.841	0.855	0.735	0.845	0.855	0.688
XGBoost - FastAI - DI_PCA	0.089	0.763	0.775	0.781	0.845	0.854	0.734	0.844	0.854	0.696
RF - GATE - DI_IN	0.087	0.693	0.716	0.751	0.860	0.861	0.769	0.860	0.861	0.748
ET - GATE - DI_IN	0.087	0.693	0.706	0.756	0.838	0.805	0.788	0.838	0.810	0.779
XGBoost - FastAI - DI_TSNE	0.085	0.760	0.775	0.783	0.845	0.855	0.746	0.844	0.854	0.707
RF - CE - DI_IN	0.079	0.690	0.721	0.775	0.860	0.861	0.766	0.860	0.861	0.744
ET - CE - DI_IN	0.079	0.690	0.721	0.772	0.840	0.810	0.790	0.837	0.817	0.779
ET - CE - DI_PCA	0.079	0.738	0.753	0.800	0.847	0.824	0.791	0.842	0.827	0.781
RF - CE - DI_PCA	0.079	0.743	0.759	0.799	0.863	0.862	0.767	0.863	0.862	0.746
ET - GATE - DI_TSNE	0.079	0.742	0.751	0.791	0.842	0.835	0.789	0.843	0.836	0.782
RF - GATE - DI_TSNE	0.078	0.752	0.761	0.786	0.863	0.861	0.770	0.863	0.861	0.751
ET - GATE - DI_PCA	0.077	0.739	0.745	0.775	0.844	0.831	0.790	0.844	0.834	0.783
RF - GATE - DI_PCA	0.077	0.745	0.758	0.770	**0.864**	0.861	0.771	0.863	0.861	0.751
ET - CE - DI_TSNE	0.076	0.736	0.756	0.788	0.844	0.829	0.790	0.843	0.832	0.780
RF - CE - DI_TSNE	0.076	0.745	0.760	0.784	0.862	0.861	0.767	0.863	0.861	0.746
XGBoost - GATE - DI_IN	0.074	0.693	0.715	0.779	0.841	0.854	0.737	0.842	0.853	0.692
XGBoost - CE - DI_IN	0.065	0.689	0.726	0.777	0.842	0.854	0.735	0.842	0.854	0.689
XGBoost - CE - DI_PCA	0.065	0.742	0.759	0.794	0.842	0.854	0.743	0.845	0.854	0.696
XGBoost - GATE - DI_TSNE	0.065	0.751	0.762	0.798	0.843	0.853	0.738	0.844	0.853	0.701
XGBoost - GATE - DI_PCA	0.064	0.743	0.759	0.794	0.844	0.853	0.736	0.845	0.853	0.699
XGBoost - CE - DI_TSNE	0.063	0.744	0.761	0.782	0.844	0.854	0.752	0.844	0.853	0.698

- Alternatively, the meta-classifier may be i) a simple majority voter (*vot*), ii) an aggregator of probabilities e.g., average (*avg*), or iii) a selection of the most confident of the three learners (*conf*, i.e., the output of the ensemble is the output of the most confident of the three base-level classifiers [92]).

6.3. Experiment execution

Experiments used the same software and hardware used in Section 4. Overall, we gathered a total of 36 combinations of base learners (3 supervised classifiers, 4 tabular DNN, 3 image DNN, 3 * 4 * 3 = 36) for creating MAND classifiers using each of the 6 meta-level strategies, for a total of 36×6 = 216 MAND classifiers. Also, we created 36×3 = 108 MAND* classifiers, letting the meta-learners *c:DT, c:GNB, c:LDA* learn from both dataset features and features *proba, maxp, ent, label* from each of the 3 base-classifiers. Noticeably, using *vot, avg, conf* meta-learners makes MAND and MAND* classifiers behave the same, thus we did not repeat these experiments. We compute metrics from previous experiments for each MAND and MAND* classifier and repeat the computation of the *disagreement* metric for groups of three classifiers

Table 5
Feature importance assigned by meta classifiers in building MAND and MAND* ensembles.

	Meta clf	dataset feats	clf1: tree-based classifier				clf2: tabular DNN				clf3: image DNN				totals			
			proba	label	maxp	ent	proba	label	maxp	ent	proba	label	maxp	ent	dataset	clf1	clf2	clf3
MAND	*c:dt*	Not used	0.221	0.375	0.053	0.308	0.009	0.002	0.003	0.002	0.012	0.006	0.005	0.003	Not used	0.957	0.017	0.026
	c:lda		0.403	0.064	0.052	0.090	0.137	0.021	0.017	0.027	0.115	0.026	0.021	0.027		0.609	0.202	0.188
	c:gnb		0.356	0.142	0.052	0.138	0.074	0.026	0.017	0.030	0.073	0.036	0.018	0.037		0.689	0.147	0.164
MAND*	*c:dt*	0.855	0.054	0.034	0.009	0.019	0.013	0.003	0.001	0.001	0.005	0.002	0.002	0.002	0.855	0.117	0.019	0.010
	c:lda	0.435	0.117	0.110	0.031	0.053	0.108	0.040	0.015	0.032	0.025	0.013	0.003	0.019	0.435	0.311	0.194	0.060
	c:gnb	0.541	0.105	0.039	0.023	0.041	0.074	0.030	0.014	0.028	0.051	0.017	0.017	0.019	0.541	0.208	0.147	0.103

instead of couples. Also, we compute the importance each feature has in building *c:DT, c:GNB, c:LDA*; this applies to MAND and MAND* classifiers, which will use features differently during their learning process i.e., MAND* classifiers also use dataset features. This will allow us to explain why MAND and MAND* have specific behavior and contribute to the overall discussion.

6.4. Classification performance of mand and MAND*

Most of the results of the experiments performed in this section are summarized in Table 4. From left to right and for each combination of base learners, the table shows the disagreement between the three base classifiers (the higher, the more diverse two classifiers are) and the MCC scores averaged on all datasets using a specific meta-classifier. We painted the background of the table with a gradient of green color: the darker, the higher the average MCC. This makes evident that the highest MCC (or better classification performance) is due to using either *c:DT* or *c:LDA* as meta-classifiers: other options are not as good as these two. There are 4 distinct combinations of base classifiers and meta-classifiers that allow MAND and MAND* ensembles to reach the best MCC of 0.864 in the table, and all use RF and DI_PCA.

Base-classifiers in Table 4 are sorted from top to bottom according to a decreasing disagreement score: the expectation is that base-classifiers close to the top of the table will build MAND and MAND* classifiers that have better classification performance as they use diverse classifiers. However, this expectation is not confirmed by results, leading to Takeover 5.

Takeover 5. The disagreement metric is not a good indicator of good classification performance of ensembles. The best MAND and MAND* classifiers are simply those that use base-learners that have excellent classification performance (RF in this case).

Section 4.5 already discussed classification performance of individual classifiers, seeing that RF, FastAI, DI_TSNE, had the highest average MCCs of 0.865, 0.703, 0.729 in their respective categories (supervised, tabular DNN, image DNN). Table 4 tells us that no matter how hard we try to combine base learners and meta-classifiers, even the best MAND and MAND* classifier does not exceed the MCC of 0.865 of RF. There's more: no MAND and MAND* classifier using XGBoost exceeds 0.855, while XGBoost alone has an average MCC of 0.859; same for ET, who has an MCC of 0.855 alone, with MAND and MAND* ensembles using ET that reach at most 0.845.

Takeover 6. There is no benefit in using tabular classifiers other than tree-based ensembles. This is true for classification performance of classifiers alone, and also when crafting complex classifiers using ensembles.

Summarizing, there is no overall benefit in combining different (and diverse) classifiers when classifying tabular data. We are aware that this goes against intuitions of solid papers in the past, but we believe that these results will finally make a conclusive statement on this research direction, letting researchers focus on other topics and using our results as a baseline to speedup future experimental studies.

6.5. Importance of features for the meta-classifier

While the results presented in the previous section clearly show that ensembling is pointless in this context, we are still interested in understanding why. A useful insight is provided by the importance given to each feature by the meta-classifiers that aggregate scores of base learners through another independent classifier: *c:DT, c:LDA, c:GNB*. Table 5 quantifies the importance each feature has in learning the model of the meta-classifier for MAND and MAND* ensembles, averaged over all triples of base learners and all datasets. All ensembles use *proba, label, maxp, ent* features from each of the three classifiers: clf1) tree-based ensemble, clf2) tabular DNN, clf3) image DNN; additionally, MAND* ensembles use also dataset features. The right of the table we report totals, where we aggregate the importance given to all features related to clf1, clf2, clf3 into a unique cumulative value.

This is very helpful for identifying how the final decision of MAND and MAND* ensembles is produced. In case of MAND, the decision is mostly due to the output of the tree-based ensemble (see the column total-clf1 in the first three rows of Table 5). In this situation, the MAND will behave almost exactly as the tree-based ensembles, with a very poor likelihood of making other decisions. For MAND*, the decision is primarily taken using dataset features but, when using c: *lda* and c:*gnb* as meta-classifiers, it accounts also for results of clf1, clf2 and, to a lesser extent, clf3, the image DNN. On paper, this gives fair importance to all base learners and dataset features, setting the ensemble for success. However, as already discussed in the previous section and in Table 4, MAND* ensembles never achieve peak classification performance. For these ensembles, we can conclude that diversity in the base-learning process is more confusing than helpful as it leads to more misclassifications instead of lowering them.

7. Threats to validity and reproducibility

Internal validity is concerned with factors that may have influenced the results, but they have not been thoroughly considered in the study. A first threat is due to the usage of public datasets, since a poor choice of datasets may invalidate the results [93]. To mitigate this risk, we used many datasets (23) for intrusion and error detection, that have been created from diverse systems by diverse authors. These are very well-known datasets, and they are largely studied and used by the scientific community. The usage of many dataset is a key to mitigate individual bias and guarantee validity of our results.

Another concern is that classifiers have hyperparameters whose tuning critically affects results. To find the appropriate tuning, we exercised sensitivity analyses for the main parameters of DNN classifiers considered in this study plus GNB and LR classifiers, using the *HyperOpt* [84] library for the latter algorithm. Third, each classifier may encounter a wide variety of problems when learning a model for each dataset during training (e.g., under/overfitting, poor quality of features, feature selection to leave out noisy features). We believe that these events are mostly situational and do not have a noticeable impact when looking at the detection performance of the same classifier or ensemble of classifiers over a span of many datasets. Another concern is the potential lack of computational resources. The proper configuration and execution of classifiers is unavoidably linked to the availability of sufficient computational resources. The resource usage of the classifiers has been properly monitored, so that we can guarantee that exhaustion of resources, especially RAM and GPU memory, was never experienced during our experiments.

An important threat to the internal validity relies upon the **fairness of the evaluation** due to possible biases that authors of the paper may have due to conflict of interest that may shift the preference from a classifier to another, or cherry-picking specific results to support a personal belief. Authors of the paper are not authors nor know any of the creators of the classifiers presented herein, and use them as independent researchers. Thus, there is no conflict of interest and there will be no reason for us to favor a classifier or another. Results and discussions presented here are solely based on our observations over a span of many years in different scenarios and research projects, and climax here to help the scientific community grow. Also, our observations are in no way intended to discredit others' work.

External validity. We cannot claim the validity of this study beyond anomaly, error and intrusion detection for tabular datasets. However, the methodology applied in this study suits any binary and multi-class classification problem without really relying on any assumption that is specific to error and intrusion detectors. Therefore, we expect our results to generalize well to other datasets and case studies for which the classification of tabular data is a relevant task. An exception may be

represented by microarray datasets [94], where the number of features exceeds the number of data points.

The usage of public data and public tools to run classifiers was a prerequisite of our analysis to allow **reproducibility** and to rely on proven-in-use data. We publicly shared scripts, methodologies, and all metric scores, allowing any researcher or practitioner to repeat the experiments. We do not use any custom or private dataset: all datasets are referenced in the papers, and all code is available at https://github.com/tommyippoz/MANDALA [95]. The folder "tests" of the GitHub above contains all scripts and data needed to reproduce results presented and discussed in the paper.

8. Concluding remarks

Whereas the popularity of image classification and robustness to adversarial attacks has never been higher, the classification of (big) tabular data gathered after monitoring activities of ICT systems is still one of the most important challenges that still lacks key contributions. One of the biggest problems is the setup of anomaly-based error and intrusion detection in critical systems, which is a tedious, time-consuming, and difficult process that may not even reach satisfactory classification performance, making companies and stakeholders reluctant to adopt classifiers at all.

Our study summarized the main features of anomaly-based error and intrusion detectors for critical systems, and conducted experimental campaigns that analyzed, compared, and discussed the performance of different classifiers on a total of 23 public datasets, even ensembling different classifiers seeking an improvement of classification performance compared to baselines. We used a wide set of classifiers, ranging from traditional supervised classifiers to tabular DNNs to image DNNs that we applied after transforming tabular data into images.

The conclusions of our study, highlighted by Takeovers 1 to 6 throughout the paper, allow debunking the problem of crafting a supervised anomaly-based error and intrusion detector for a given system. Classifiers using Decision Trees (tree-based ensembles) as bagging and boosting classifiers outperform DNNs, are faster to train, and have a very small set of hyperparameters that are easy to understand and tune. Ensembling tree-based classifiers and DNNs did not lead to any improvement in classification performance, nor did the usage of image DNNs after data transformation. These observations hold for all datasets used in this study, and there is no reason to think that our conclusions will not hold for other tabular (big) data classification problems.

We believe that this study could provide a solid baseline upon which practitioners willing to solve a specific tabular data classification problem could rely to guide the design and implementation of their solution. Anomaly-based error and intrusion detection in tabular data should be carried out through tree-based ensembles disregarding DNNs – either tabular DNNs or image DNNs exercised on a transformed tabular input -, saving key time when setting up these classifiers. Also, the MANDALA public library [95] can be used by any practitioner to cross-check if the findings of the paper also apply to their case study.

CRediT authorship contribution statement

Tommaso Zoppi: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Data curation, Conceptualization. **Stefano Gazzini:** Software, Methodology. **Andrea Ceccarelli:** Writing – review & editing, Validation, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

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

Data availability

datasets are publicly available (not ours) and the code is publicly available on the GitHub repository linked in the paper

Acknowledgements

This work was supported in part by the B53D23012930006 PRIN 2022 project FLEGREA, the 202297YF75 PRIN 2022 project S2, and by the project P2022K7ERB PRIN PNRR 2022 BREADCRUMBS and by the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU.

References

[1] M. Gil, M. Albert, J. Fons, V. Pelechano, Designing human-in-the-loop autonomous Cyber-Physical Systems, Int. J. Hum. Comput. Stud. 130 (2019) 21–39, https://doi.org/10.1016/j.ijhcs.2019.04.006.
[2] I.F. Akyildiz, A. Kak, The Internet of Space Things/CubeSats: a ubiquitous cyber-physical system for the connected world, Comput. Networks 150 (Feb. 2019) 134–149, https://doi.org/10.1016/j.comnet.2018.12.017.
[3] G. Wang, L. Zhang, W. Xu, What can we learn from four years of data center hardware failures?, in: 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) IEEE, 2017, pp. 25–36, https://doi.org/10.1109/DSN.2017.26.
[4] G. Jesus, A. Casimiro, A. Oliveira, Using machine learning for dependable outlier detection in environmental monitoring systems, ACM Transactions on Cyber-Phys. Syst. 5 (3) (2021) 1–30, https://doi.org/10.1145/3445812.
[5] J. Zhang, B. Zhang, N. Zhang, C. Wang, Y. Chen, A novel robust event-triggered fault tolerant automatic steering control approach of autonomous land vehicles under in-vehicle network delay, Int. J. Robust Nonlinear Control 31 (7) (2021) 2436–2464, https://doi.org/10.1002/rnc.5393.
[6] C. Abbey, et al., Powering through the storm: microgrids operation for more efficient disaster recovery, IEEE Power Energ. Mag. 12 (3) (2014) 67–76, https://doi.org/10.1109/MPE.2014.2301514.
[7] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Trans. Dependable Secure Comput. 1 (1) (2004) 11–33, https://doi.org/10.1109/TDSC.2004.2.
[8] Z. Xu, J.H. Saleh, Machine learning for reliability engineering and safety applications: review of current status and future opportunities, Reliab. Eng. Syst. Saf. 211 (2021) 107530, https://doi.org/10.1016/j.ress.2021.107530.
[9] L. Grinsztajn, E. Oyallon, G. Varoquaux, Why do tree-based models still outperform deep learning on typical tabular data? Adv. Neural. Inf. Process. Syst. 35 (2022) 507–520.
[10] D.-T. Nguyen, K.-H. Le, The robust scheme for intrusion detection system in Internet of Things, Internet. Things 24 (2023) 100999, https://doi.org/10.1016/j.iot.2023.100999.
[11] S. Ö. Arik, T. Pfister, TabNet: attentive Interpretable Tabular Learning, in: Proceedings of the AAAI Conference on Artificial Intelligence 35, 2021, pp. 6679–6687, https://doi.org/10.1609/aaai.v35i8.16826.
[12] S. Popov, S. Morozov, A. Babenko, Neural oblivious decision ensembles for deep learning on tabular data, in: International Conference on Learning Representations, 2020 [Online]. Available: https://openreview.net/forum?id=r1eiu2VtwH.
[13] J. Howard, S. Gugger, Fastai: a layered API for deep learning, Information 11 (2) (2020) 108, https://doi.org/10.3390/info11020108.
[14] D. Ardagna, C. Cappiello, W. Samá, M. Vitali, Context-aware data quality assessment for big data, Future Generat. Comput. Syst. 89 (2018) 548–562, https://doi.org/10.1016/j.future.2018.07.014.
[15] M. Molan, A. Borghesi, D. Cesarini, L. Benini, A. Bartolini, RUAD: unsupervised anomaly detection in HPC systems, Future Generat. Comput. Syst. 141 (2023) 542–554, https://doi.org/10.1016/j.future.2022.12.001.
[16] S. Leroux, P. Simoens, Sparse random neural networks for online anomaly detection on sensor nodes, Future Generat. Comput. Syst. 144 (2023) 327–343, https://doi.org/10.1016/j.future.2022.12.028.
[17] R. Shwartz-Ziv, A. Armon, Tabular data: deep learning is not all you need, Information Fusion 81 (2022) 84–90, https://doi.org/10.1016/j.inffus.2021.11.011.
[18] Y. Zhu, et al., Converting tabular data into images for deep learning with convolutional neural networks, Sci. Rep. 11 (1) (2021) 11325, https://doi.org/10.1038/s41598-021-90923-y.
[19] A. Sharma, E. Vans, D. Shigemizu, K.A. Boroevich, T. Tsunoda, DeepInsight: a methodology to transform a non-image data to an image for convolution neural network architecture, Sci. Rep. 9 (1) (2019) 11399, https://doi.org/10.1038/s41598-019-47765-6.
[20] S. González, S. García, J. Del Ser, L. Rokach, F. Herrera, A practical tutorial on bagging and boosting based ensembles for machine learning: algorithms, software tools, performance study, practical perspectives and opportunities, Information Fusion 64 (2020) 205–237, https://doi.org/10.1016/j.inffus.2020.07.007.

[21] T. Zoppi, M. Gharib, M. Atif, A. Bondavalli, Meta-learning to improve unsupervised intrusion detection in cyber-physical systems, ACM Transact. Cyber-Phys. Syst. 5 (4) (2021), https://doi.org/10.1145/3467470.

[22] D.H. Wolpert, Stacked generalization, Neural Netw. 5 (2) (1992) 241–259, https://doi.org/10.1016/S0893-6080(05)80023-1.

[23] Y. Freund, Boosting a weak learning algorithm by majority, Inf. Comput. 121 (2) (1995) 256–285, https://doi.org/10.1006/inco.1995.1136.

[24] T. Zoppi, A. Ceccarelli, A. Bondavalli, MADneSs: a multi-layer anomaly detection framework for complex dynamic systems, IEEE Trans. Dependable Secure Comput. 18 (2) (2021), https://doi.org/10.1109/TDSC.2019.2908366.

[25] A. Khraisat, I. Gondal, P. Vamplew, J. Kamruzzaman, Survey of intrusion detection systems: techniques, datasets and challenges, Cybersecur. 2 (1) (2019) 20, https://doi.org/10.1186/s42400-019-0038-7.

[26] H. Rajadurai, U.D. Gandhi, A stacked ensemble learning model for intrusion detection in wireless network, Neural. Comput. Appl. 34 (18) (2022) 15387–15395, https://doi.org/10.1007/s00521-020-04986-5.

[27] R.H. Randhawa, N. Aslam, M. Alauthman, M. Khalid, H. Rafiq, Deep reinforcement learning based Evasion Generative Adversarial Network for botnet detection, Future Generat. Comput. Syst. 150 (2024) 294–302, https://doi.org/10.1016/j.future.2023.09.011.

[28] C. Pham, Z. Estrada, P. Cao, Z. Kalbarczyk, R.K. Iyer, Reliability and security monitoring of virtual machines using hardware architectural invariants, in: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, IEEE, 2014, pp. 13–24, https://doi.org/10.1109/DSN.2014.19.

[29] E. De Giovanni, A.A. Valdes, M. Peon-Quiros, A. Aminifar, D. Atienza, Real-time personalized atrial fibrillation prediction on multi-core wearable sensors, IEEE Trans. Emerg. Top. Comput. 9 (4) (2021) 1654–1666, https://doi.org/10.1109/TETC.2020.3014847.

[30] G. Hu, Y. Guo, G. Wei, L. Abualigah, Genghis Khan shark optimizer: a novel nature-inspired algorithm for engineering optimization, Adv. Eng. Inf. 58 (2023) 102210, https://doi.org/10.1016/J.AEI.2023.102210.

[31] J.O. Agushaka, A.E. Ezugwu, L. Abualigah, Dwarf mongoose optimization algorithm, Comput. Methods Appl. Mech. Eng. 391 (2022) 114570, https://doi.org/10.1016/J.CMA.2022.114570.

[32] L. Le, A. Patterson, M. White, Supervised autoencoders: improving generalization performance with unsupervised regularizers, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc., 2018 [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/2a38a4a9316c49e5a833517c45d31070-Paper.pdf.

[33] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, Z. Chen, SySeVR: a framework for using deep learning to detect software vulnerabilities, IEEE Trans. Dependable Secure Comput. 19 (4) (2022) 2244–2258, https://doi.org/10.1109/TDSC.2021.3051525.

[34] G. Li, J.J. Jung, Deep learning for anomaly detection in multivariate time series: approaches, applications, and challenges, Informat. Fusion 91 (2023) 93–102, https://doi.org/10.1016/j.inffus.2022.10.008.

[35] L. Breiman, Random forests, Mach Learn 45 (1) (2001) 5–32, https://doi.org/10.1023/A:1010933404324.

[36] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, Mach Learn 63 (1) (2006) 3–42, https://doi.org/10.1007/s10994-006-6226-1.

[37] T. Chen, C. Guestrin, XGBoost: a Scalable Tree Boosting System, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2016, pp. 785–794, https://doi.org/10.1145/2939672.2939785.

[38] W.J. Krzanowski, T.C. Bailey, D. Partridge, J.E. Fieldsend, R.M. Everson, V. Schetinin, Confidence in classification: a bayesian approach, J Classif 23 (2) (2006) 199–220, https://doi.org/10.1007/s00357-006-0013-3.

[39] Y. Liao, V.R. Vemuri, Use of K-Nearest Neighbor classifier for intrusion detection, Comput. Secur. 21 (5) (2002) 439–448, https://doi.org/10.1016/S0167-4048(02)00514-X.

[40] M.A. Souza, R. Sabourin, G.D.C. Cavalcanti, R.M.O. Cruz, A dynamic multiple classifier system using graph neural network for high dimensional overlapped data, Informat. Fusion 103 (2024) 102145, https://doi.org/10.1016/j.inffus.2023.102145.

[41] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444, https://doi.org/10.1038/nature14539.

[42] L. van der Maaten, G. Hinton, Visualizing Data using t-SNE, J. Mach. Learn Res. 9 (86) (2008) 2579–2605 [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html.

[43] R. Vilalta, C. Giraud-Carrier, P. Brazdil, Meta-Learning - Concepts and Techniques. Data Mining and Knowledge Discovery Handbook, Springer US, Boston, MA, 2009, pp. 717–731, https://doi.org/10.1007/978-0-387-09823-4_36.

[44] N.O. Nikitin, et al., Automated evolutionary approach for the design of composite machine learning pipelines, Future Generat. Comput. Syst. 127 (2022) 109–125, https://doi.org/10.1016/j.future.2021.08.022.

[45] T. Windeatt, Diversity measures for multiple classifier system analysis and design, Informat. Fusion 6 (1) (2005) 21–36, https://doi.org/10.1016/j.inffus.2004.04.002.

[46] E.K. Tang, P.N. Suganthan, X. Yao, An analysis of diversity measures, Mach Learn 65 (1) (2006) 247–271, https://doi.org/10.1007/s10994-006-9449-2.

[47] L.I. Kuncheva, C.J. Whitaker, Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy, Mach Learn 51 (2) (2003) 181–207, https://doi.org/10.1023/A:1022859003006.

[48] Z. Gong, P. Zhong, W. Hu, Diversity in machine learning, IEEE Access 7 (2019) 64323–64350, https://doi.org/10.1109/ACCESS.2019.2917620.

[49] L. Breiman, Bagging predictors, Mach Learn 24 (2) (1996) 123–140, https://doi.org/10.1007/BF00058655.

[50] R.E. Schapire, The strength of weak learnability, Mach Learn 5 (2) (1990) 197–227, https://doi.org/10.1007/BF00116037.

[51] C. Lemke, M. Budka, B. Gabrys, Metalearning: a survey of trends and technologies, Artif. Intell. Rev. 44 (1) (2015) 117–130, https://doi.org/10.1007/s10462-013-9406-y.

[52] S. Džeroski, B. Ženko, Is combining classifiers with stacking better than selecting the best one? Mach Learn 54 (3) (2004) 255–273, https://doi.org/10.1023/B:MACH.0000015881.36452.6e.

[53] T. Zoppi and A. Ceccarelli, "Prepare for trouble and make it double. supervised and unsupervised stacking for anomalybased intrusion detection," *arXiv*. 2022. https://doi.org/10.48550/arxiv.2202.13611.

[54] K.M. Ting, I.H. Witten, Issues in Stacked Generalization, J. Artificial Intelligence Res. 10 (1999) 271–289, https://doi.org/10.1613/jair.594.

[55] Sina Mohseni, Mandar Pitale, Vasu Singh, Zhangyang Wang, Practical solutions for machine learning safety in autonomous vehicles. **SafeAI Workshop @ AAAI**, 2019.

[56] X. Ma, J. Zhu, Z. Lin, S. Chen, Y. Qin, A state-of-the-art survey on solving non-IID data in Federated Learning, Future Generat. Comput. Syst. 135 (Oct. 2022) 244–258, https://doi.org/10.1016/j.future.2022.05.003.

[57] J. Luo, Y. Quan, S. Xu, Robust-GBDT: A Novel Gradient Boosting Model for Noise-Robust Classification, 2023 [Online]. Available: http://arxiv.org/abs/2310.05067.

[58] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Trans. Evol. Comput. 1 (1) (1997) 67–82, https://doi.org/10.1109/4235.585893.

[59] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, A. Hotho, A survey of network-based intrusion detection data sets, Comput. Secur. 86 (2019) 147–167, https://doi.org/10.1016/j.cose.2019.06.005.

[60] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho, "Flow-based benchmark data sets for intrusion detection," 2017.

[61] A.H. Lashkari, A.F.A. Kadir, L. Taheri, A.A. Ghorbani, Toward developing a systematic approach to generate benchmark android malware datasets and classification, in: 2018 International Carnahan Conference on Security Technology (ICCST), IEEE, 2018, pp. 1–7, https://doi.org/10.1109/CCST.2018.8585560.

[62] B.M.M.Y.M.Y.B.D.A. Meidan Yair, A. Shabtai, Detection_Of_Iot_Botnet_Attacks_N_BaioT, 2018.

[63] W. Haider, J. Hu, J. Slay, B.P. Turnbull, Y. Xie, Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling, J. Netw. Comput. Appl. 87 (2017) 185–192, https://doi.org/10.1016/j.jnca.2017.03.018.

[64] I. Sharafaldin, A. Habibi Lashkari, A.A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization, in: Proceedings of the 4th International Conference on Information Systems Security and Privacy, SCITEPRESS - Science and Technology Publications, 2018, pp. 108–116, https://doi.org/10.5220/0006639801080116.

[65] H. Kang, D.H. Ahn, G.M. Lee, J. Do Yoo, K.H. Park, H.K. Kim, IoT network intrusion dataset, IEEE Dataport (2019), https://doi.org/10.21227/q70p-q449.

[66] A. Shiravi, H. Shiravi, M. Tavallaee, A.A. Ghorbani, Toward developing a systematic approach to generate benchmark datasets for intrusion detection, Comput. Secur. 31 (3) (2012) 357–374, https://doi.org/10.1016/j.cose.2011.12.012.

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

[68] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, R. Therón, UGR'16: a new dataset for the evaluation of cyclostationarity-based network IDSs, Comput. Secur. 73 (2018) 411–424, https://doi.org/10.1016/j.cose.2017.11.004.

[69] N. Moustafa, J. Slay, UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set, in: 2015 Military Communications and Information Systems Conference (MilCIS), IEEE, 2015, pp. 1–6, https://doi.org/10.1109/MilCIS.2015.7348942.

[70] BackBlaze, "BackBlaze HDD Data," https://www.backblaze.com/cloud-storage/resources/hard-drive-test-data [accessed: June 7th, 2024].

[71] Baidu Inc, "Baidu HDD - Baidu SMART Dataset for Seagate ST31000524NS Drive Model," 2022. https://www.kaggle.com/datasets/drtycoon/hdds-dataset-baidu-inc.

[72] W.C.S.Y.J.-H. Shin Hyeok-Ki, Lee, B.-G. Min, HAI Security Datasets, 2023 [Online]. Available: https://github.com/icsdataset/hai.

[73] H.-K. Shin, W. Lee, J.-H. Yun, H. Kim, HAI 1.0: hIL-based Augmented ICS Security Dataset, in: 13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20), USENIX Association, 2020 [Online]. Available: https://www.usenix.org/conference/cset20/presentation/shin.

[74] N. Davari, B. Veloso, R.P. Ribeiro, P.M. Pereira, J. Gama, Predictive maintenance based on anomaly detection using deep learning for air production unit in the railway industry, in: 2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA), IEEE, 2021, pp. 1–10, https://doi.org/10.1109/DSAA53316.2021.9564181.

[75] APS Failure at Scania Trucks, 2017.

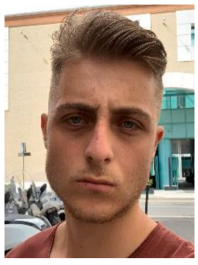
[76] T. Zoppi, G. Merlino, A. Ceccarelli, A. Puliafito, A. Bondavalli, Anomaly Detectors for Self-Aware Edge and IoT Devices, in: IEEE International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2023, p. 23AD.

[77] A. Agarwal, Machine Failure Prediction, Kaggle (2018) [Online]. Available: https://kaggle.com/competitions/machine-failure-prediction.

[78] M.A. Marins, F.M.L. Ribeiro, S.L. Netto, E.A.B. da Silva, Improved similarity-based modeling for the classification of rotating-machine failures, J. Franklin Inst. 355 (4) (2018) 1913–1930, https://doi.org/10.1016/j.jfranklin.2017.07.038.

[79] S.R. Safavian, D. Landgrebe, A survey of decision tree classifier methodology, IEEE Trans. Syst. Man Cybern. 21 (3) (1991) 660–674, https://doi.org/10.1109/21.97458.
[80] T. Chen, C. Guestrin, XGBoost, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2016, pp. 785–794, https://doi.org/10.1145/2939672.2939785.
[81] L. Jiang, D. Wang, Z. Cai, X. Yan, Survey of Improving Naive Bayes for Classification, 2007, pp. 134–145, https://doi.org/10.1007/978-3-540-73871-8_14.
[82] J. Friedman, T. Hastie, R. Tibshirani, Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors, The Annals of Statistics 28 (2) (2000), https://doi.org/10.1214/aos/1016218223.
[83] P. Xanthopoulos, P.M. Pardalos, T.B. Trafalis, Robust Data Mining, Springer New York, New York, NY, 2013, https://doi.org/10.1007/978-1-4419-9878-1.
[84] B. Komer, J. Bergstra, C. Eliasmith, Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. ICML Workshop On AutoML, 2014, p. 50.
[85] M. Joseph, H. Raj, GANDALF: Gated Adaptive Network for Deep Automated Learning of Features, 2022.
[86] A. Yang, MNIST Tutorial by Kaggle, 2020, https://www.kaggle.com/code/amyjang/tensorflow-mnist-cnn-tutorial.
[87] D. Chicco, G. Jurman, The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation, Bmc Genomics [Electronic Resource] 21 (1) (2020) 6, https://doi.org/10.1186/s12864-019-6413-7.
[88] K.H. Brodersen, C.S. Ong, K.E. Stephan, J.M. Buhmann, The balanced accuracy and its posterior distribution, in: 2010 20th International Conference on Pattern Recognition, IEEE, 2010, pp. 3121–3124, https://doi.org/10.1109/ICPR.2010.764.
[89] M. Maalouf, M. Siddiqi, Weighted logistic regression for large-scale imbalanced and rare events data, Knowl Based Syst 59 (2014) 142–148, https://doi.org/10.1016/J.KNOSYS.2014.01.012.
[90] N. Burkart, M.F. Huber, A Survey on the Explainability of Supervised Machine Learning, J. Artificial Intelligence Res. 70 (2021) 245–317, https://doi.org/10.1613/jair.1.12228.
[91] A. Biondi, F. Nesti, G. Cicero, D. Casini, G. Buttazzo, A safe, secure, and predictable software architecture for deep learning in safety-critical systems, IEEE Embed Syst Lett 12 (3) (2020) 78–82, https://doi.org/10.1109/LES.2019.2953253.
[92] C. Ferri, P. Flach, J. Hernández-Orallo, Delegating classifiers, in: Twenty-first international conference on Machine learning - ICML '04, ACM Press, New York, New York, USA, 2004, p. 37, https://doi.org/10.1145/1015330.1015395.
[93] M. Catillo, A. Pecchia, M. Rak, U. Villano, Demystifying the role of public intrusion datasets: a replication study of DoS network traffic data, Comput. Secur. 108 (2021) 102341, https://doi.org/10.1016/j.cose.2021.102341.
[94] V. Bolón-Canedo, N. Sánchez-Maroño, A. Alonso-Betanzos, J.M. Benítez, F. Herrera, A review of microarray datasets and applied feature selection methods, Inf Sci (N Y) 282 (2014) 111–135, https://doi.org/10.1016/j.ins.2014.05.042.
[95] T. Zoppi, MANDALA GitHub Repository, 2024. https://github.com/tommyippoz/MANDALA.

Tommaso Zoppi: received his Ph.D. in Computer Science from the University of Firenze (Italy) in 2018. He is currently a tenure-tracked Research associate (RTD-B) at the University of Trento, mainly working on critical systems engineering and the implications of the usage of machine learning in these systems. Throughout years, he participated in a wide variety of regional, national, EU, and industrial research projects in the broad domain of architectures, verification and validation of critical systems. He serves as TPC member in various international conferences and as a reviewer of top-rated journals, and he co-authored more than 40 papers to date.



Stefano Gazzini: is a computer science student with a strong passion in data and artificial intelligence. During his studies he has delved into the main topics of the discipline, such as computer architecture, databases and programming but in the final thesishe has explored the field of artificial intelligence with a project whose main goal was totest the performance of several algorithms in computing tabular data.

Andrea Ceccarelli: received the Ph.D. degree in informatics and automation engineering from the University of Florence, Florence, Italy, in 2012. He is currently an Associate Professor of computer science with the University of Florence. His research interests include the design, monitoring, and evaluation of dependable and secure systems, with a preference for experimental approaches, and his scientific activities originated more than 120 papers. He is regularly involved in the TPC of International Conferences in the domain of dependability and reliability engineering, and he has been the TPC CoChair of LADC and SRDS. He has been involved in multiple research projects and he is currently leading his unit in the MUR projects BREADCRUMBS and FLEGREA. He is a member of the IFIP WG 10.4 on "Dependable Computing and Fault-Tolerance."