Data Science and Engineering
https://doi.org/10.1007/s41019-024-00264-9

RESEARCH PAPER

A Strategy for Predicting the Performance of Supervised and Unsupervised Tabular Data Classifiers

Tommaso Zoppi[1] · **Andrea Ceccarelli**[1] · **Andrea Bondavalli**[1]

Received: 10 April 2024 / Revised: 11 September 2024 / Accepted: 13 September 2024

Abstract
Machine Learning algorithms that perform classification are increasingly been adopted in Information and Communication Technology (ICT) systems and infrastructures due to their capability to profile their expected behavior and detect anomalies due to ongoing errors or intrusions. Deploying a classifier for a given system requires conducting comparison and sensitivity analyses that are time-consuming, require domain expertise, and may even not achieve satisfactory classification performance, resulting in a waste of money and time for practitioners and stakeholders. This paper predicts the expected performance of classifiers without needing to select, craft, exercise, or compare them, requiring minimal expertise and machinery. Should classification performance be predicted worse than expectations, the users could focus on improving data quality and monitoring systems instead of wasting time in exercising classifiers, saving key time and money. The prediction strategy uses scores of feature rankers, which are processed by regressors to predict metrics such as Matthews Correlation Coefficient (MCC) and Area Under ROC-Curve (AUC) for quantifying classification performance. We validate our prediction strategy through a massive experimental analysis using up to 12 feature rankers that process features from 23 public datasets, creating additional variants in the process and exercising supervised and unsupervised classifiers. Our findings show that it is possible to predict the value of performance metrics for supervised or unsupervised classifiers with a mean average error (MAE) of residuals lower than 0.1 for many classification tasks. The predictors are publicly available in a Python library whose usage is straightforward and does not require domain-specific skill or expertise.

Keywords Feature ranking · Anomaly detection · Classification performance · Machine learning · Error detection

1 Introduction

Nowadays the paradigm of Cyber-Physical Systems (CPSs) [27] 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 [25, 56, 65],Saied, Guirguis and Madbouly, 2024). 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, 87]. 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 significantly 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].

✉ Tommaso Zoppi
tommaso.zoppi@unifi.it

Andrea Ceccarelli
andrea.ceccarelli@unifi.it

Andrea Bondavalli
bondavalli@unifi.it

[1] Department of Mathematics and Informatics, University of Florence, Viale Morgagni 65, 50142 Florence, Italy

Published online: 28 October 2024

1.1 Anomaly-Based Classifiers

From a general standpoint, anomaly-based classifiers (classifiers from now on) (Chandola, Banerjee and Kumar, 2009)"*identify patterns that do not conform to a well-defined notion of normal behavior*". During training, those classifiers learn a model that allows them to output either a positive or a negative class depending on feature values. Later, the classifier uses this model to label novel data points either as negative or positive i.e., corresponding to a normal state of the system, or hinting to ongoing errors, attacks, or performance anomalies in general. Their classification performance is typically expressed using metrics [45] that combine correct classifications—True Positives (TPs) and Negatives (TNs) – which represent the desired outcome, and misclassifications as False Positives (FPs) and False Negatives (FNs), to be minimized. The classification problem may include many classes, of which one represents a normal behavior, and others represent different types of anomalous behaviors. In this case, classifiers should be able to distinguish between normal behavior and each specific class of anomaly: these are referred to as multi-class classifiers e.g., intrusion detectors that aim at identifying attacks but also ad distinguishing between different categories of attacks. With multi-class classification problems, evaluation metrics have to be adapted to suit the dimensionality of the problem [17].

Different classifiers may achieve different classification performance when dealing with the same task in a specific system. Supervised classifiers [19, 30, 52] were proven to achieve excellent detection performance in many domains: they learn their model using data points collected i) during normal operation of the system, ii) when errors, attacks or failures activate, and labelled accordingly. 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 [32, 60]. Alternatively, unsupervised classifiers [29, 57, 96] do not require labelled data for training; thus, they are applicable whenever labels are not available, too expensive to derive, or when dealing with evolving systems or unknown threats [73, 94].

1.2 Engineering Classifiers

Building and deploying a classifier for a given system requires many steps [6] which include, but are not limited to feature selection, feature engineering [12], classifier selection, analyses of hyper-parameters through grid, random or gradient-descend searches, and comparison of metric scores achieved by different classifiers. In the vast majority of cases, the classification performance of classifiers strictly depends on the number and/or the quality of monitored system indicators, which constitute the baseline to create features for training classifiers [83].

Unfortunately, it may happen that even that best classifier does not achieve satisfactory classification performance for the problem at hand, ending up having no practical use. When this happens, the only available option is to rework the monitoring system or the feature engineering process, providing classifiers with different—and more informative—features before starting the selection process from scratch again. From an engineering or stakeholder standpoint, this event is highly detrimental: the time, resources, and expertise that was devoted to creating such a classifier at the first stage is wasted and represents an economic loss for the company that was willing to deploy the classifier in their application.

1.3 Motivation

It would be very helpful to know in advance if features – or the monitoring system—are "good enough" to model classifiers with satisfactory classification performance. This would open the possibility of assessing the quality of input data for learning a classification model that applies to supervised or unsupervised, binary, or multi-class classification tasks in any domain in which tabular data is involved. Ideally, this process should be completely decoupled from the classifier that will be selected at a later stage, and it should provide fast (i.e., negligible overhead) feedback to be used in the early stages of the data analysis task. To accomplish that, the classification performance prediction strategy should primarily rely on data that is already available through traditional data analysis, pre-processing, and feature engineering techniques. Conversely, implementing the strategy could require researchers or practitioners to commit a major time and resource overhead which may be deemed unfeasible.

With the prediction at hand, the researcher or the practitioner should decide to invest time in building the classifier only if the predicted classification performance is satisfactory. Conversely, engineering efforts should be redirected to improving the quality or number of features that—at the current state – do not allow for building adequate classifiers.

1.4 Technical Contribution

This paper introduces a strategy to predict the expected classification performance of a task on a given system or dataset. Our prediction strategy quantifies the goodness of available features by exercising feature rankers, feeding their results to a regressor that predicts the numeric

value of a classification metric. The regressor is an ML algorithm that outputs a continuous numeric label instead of a categorical label (as classifiers do). The output of the regressor constitutes an easy-to-interpret prediction of the classification performance that can be expected from classifiers trained using a specific dataset. For the sake of brevity, we could not report experimental results in predicting all metrics available in the literature. We show that it is possible to predict values of classification metrics of Area Under ROC Curve (AUC) and Matthews Correlation Coefficient (MCC) with minimal error and in a few minutes at most, even when dealing with datasets containing hundreds of thousands of data points and many features. Our experimental evaluation mainly focuses on typical applications of anomaly-based classifiers such as intrusion detection, error detection, and hardware failure prediction, but is tested also on datasets belonging to different domains. The classifiers used therein are those that are typically recommended when dealing with tabular data: the supervised tree-based ensembles as Random Forests, ExtraTrees and boosting algorithms as XGBoost and LogitBoost, and unsupervised classifiers suggested in [29, 94], namely G-Means, HBOS, Self-Organizing Maps, Isolation Forests, ODIN, SDO and FastABOD. Experiments have been executed using the FRAPPE (Feature RAnkers to Predict classification PerformancE of classifiers) framework, which is available as an open-source library on GitHub (Anonymous, no date) and as PyPI package, and includes all scripts needed to reproduce our results. Our main technical findings of the paper, which are discussed throughout the paper, are as follows.

- The prediction strategy applies to any classification problem and can quantify any existing classification metrics:
- Outputs of feature rankers can be used to predict the classification performance of supervised or unsupervised classifiers with low prediction error in a given dataset, conversely to existing similar studies.
- Our prediction strategy can be integrated into existing data analysis processes requiring little to no modifications to the existing analysis flow and quantifies the prediction in a few minutes at most.
- The FRAPPE public Python library provides a user-friendly and easy-to-use interface, requiring minimal expertise to use the findings of the paper as the library hides all the implementation details.
- The prediction strategy that is available in FRAPPE can be applied to tabular datasets from any domain, predicting the performance of unsupervised binary classifiers and supervised binary and multi-class classifiers with a mean average error (MAE) of residuals lower than 0.1 for metrics as MCC and AUC many classification tasks.

1.5 Paper Structure

The remainder of the paper is structured as follows. Section 2 reports terminology, basics, and literature related to classifiers and feature rankers, while Sect. 3 motivates the usefulness of our strategy to predict classification performance, and lists related works. Section 4 presents our prediction strategy, which is evaluated according to the experimental methodology in Sect. 5 and discussed in Sect. 6. Section 7 presents the application of the prediction strategy to many case studies, letting Sect. 8 conclude the paper and debate the limitations of this work.

2 Background and Related Works

We provide an overview of supervised and unsupervised classifiers for anomaly detection in tabular data. Then, we review techniques for feature ranking and selection and finally summarize metrics to quantify classification performance.

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 [44, 57, 95] and intrusion [38, 66, 67] 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 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 [65], system-level [62], environment [25], or application-level monitoring [28]. 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 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. More formally, a classifier *clf* first devises a mathematical model from a training dataset [13], which contains a given amount of data points. Each data point *dp* contains a set of *f* feature values, where each feature value is a floating point number dp_j with $0 \leq j < f$ and describes a specific input of the classification problem. Once the model is learned, it can be used to predict the *dp_prob* probabilities of the data point belonging to each class of the problem, of which the class with the highest probability is assigned as *dp_label* of the new data point, different from those in the training dataset. The classification performance is usually computed by applying *clf* to data points in a test dataset and computing metrics such as accuracy [45], i.e., the percentage of correct predictions of a classifier *clf* overall predictions.

The vast majority of ML algorithms that have been used for decades to tackle classification tasks are supervised classifiers (Le, Patterson and White, 2018; [30, 46, 47]). 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 such as Random Forests [16],Geurts, Ernst and Wehenkel, 2006) or XGBoost [19]), statistical techniques [39], distance-based learners [48], or neural networks (DNNs, (Le, Patterson and White, 2018; Souza et al., 2024)). DNNs are supervised classifiers that contain multiple hidden layers (deep networks) to learn different features with multiple levels of abstraction (LeCun, Bengio and Hinton, 2015). 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 object detection. However, many studies argue about their performance in classifying tabular data. For instance, Intel advocates [79] that XGBoost shows better classification performance than DNNs when dealing with tabular data. This is confirmed by [32], where authors justify the supremacy of tree-based classifiers against 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 [5] present a DNN that is optimized for tabular data and outperforms tree-based classifiers in some datasets. Similarly, the authors of Neural Oblivious Decision Ensembles (Popov, Morozov and Babenko, 2020) claim that their method is the first successful example of DNN that substantially outperforms gradient-boosting classifiers on tabular data. FastAI [35] 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.

Instead, unsupervised classifiers [29, 57] do not require any prior knowledge of the labels. This makes them suitable [94] for the detection of known and unknown errors, anomalies, and attacks, but only for binary classification problems where there is a majority class (i.e., normal data) and a minority class (i.e., anomalies). Over the years, many unsupervised algorithms have been proposed, studied, and compared to derive similarities or differences, identifying families of classifiers as clustering, density-based, angle-based, statistical, and neural networks.

2.4 Features and Feature Ranking

The baseline upon which classifiers learn how to label a data point is the features, which are defined as "individual measurable properties or characteristics of a phenomenon being observed" [13]. Each data point contains values for each feature engineered from monitored system indicators. Additional attributes, called meta-features, can be further extracted from the corresponding dataset during the process [70]. Not all features carry the same information content, whereas some of them may just represent noise. This aspect is usually quantified thanks to Feature Rankers or Selectors [40]: given a set F of features, a feature ranker fr is a function $value\_f = fr(f), f \in F, value\_f \in [0, 1]$.

The resulting $value\_f$ is a number that allows for a relative ordering of features from the most to the least relevant, allowing the selection of those that contribute the most to predict the correct label (i.e., feature selection). Typically, $value\_f$ is a $\mathbb{R}$ number, but can be easily normalized in a [0, 1] range where 0 means "no information" and 1 means "maximum information".

There are many options [40] for ranking and selecting features: *embedded* feature rankers depend on the classifier to be used at a later stage, whereas *filter-based* and *wrapper-based* feature rankers assign relevance scores to features without being specific to any classifier. Our study aims to predict the classification performance achievable with a dataset, independently of the classifier to be used afterward. As such, we will employ filter and wrapper-based feature rankers, disregarding embedded feature rankers that will unavoidably tie our prediction strategy to a specific classifier.

2.5 Filter-Based Ranking

Many strategies can rank features according to statistical filters. Statistical filter-based rankers have a common structure: they aim at computing a correlation between the feature values and the label to be predicted. As such, statistical rankers do not account for interactions between features but only account for statistical correlation or similarity between individual features and the label. Examples include but are not limited to Chi-Squared correlation [20], R-Squared correlation [55], Pearson correlation [88], ANalysis Of VAriance (ANOVA) rank [64], Spearman rank [89], Cosine Similarity [86], Information Gain [91].

2.6 Relief-Based Ranking

In addition to statistical rankers, another important family of filter-based rankers stems from the Relief [84] algorithm. This strategy for feature selection was developed thirty years ago to quantify feature relevance in a dataset using the difference in feature values between similar data points. The bigger the observed difference in a pair of data points with the same class, the lower the feature rank; alternatively, observing a small difference in feature values in similar data points makes the feature rank grow. The data deluge we witnessed in recent years combined with the polynomial time complexity required to compute Relief made this approach obsolete at least in its original formulation. Nowadays, Relief is computed on a small subset of the original dataset, and often according to variants such as Spatially Uniform RelieF (SURF [31]), MultiSURF [84], or TUned RelieF (TURF) which slightly speed up the whole feature ranking and selection process.

2.7 Wrapper-Based Ranking

Wrapper rankers demand the ranking workload from an external classifier. Feature values and labels are used to make the external classifier learn a model: the relevance each feature had in building that model is then used as feature rank. Wrapper-based rankers are usually more computationally expensive than filter-based counterparts (especially the statistic ones) as they require training a classifier. Any classifier can be used to build a wrapper-based feature ranker: commonly used rankers wrap either Random Forests [16] or Linear Regressors [23],Behera et al., 2023). Depending on the characteristics of the classification problem e.g., linear vs non-linear, the analyst may prefer going one way or another or wrapping yet another classifier for ranking.

2.8 Metrics to Evaluate Classification Performance

The detection performance of binary classifiers is primarily evaluated through a confusion matrix, thus calculating TP, TN, FP, and FN. Those four items can be aggregated into a wide variety of compound metrics [45] as False Positive Rate (FPR), Precision (P), Recall (R), F-Measure (F1-Score, or F1), F2-Score (F2), Matthews Coefficient (MCC), Accuracy (ACC), Area Under ROC Curve (AUC), which are widely adopted when calculating classification performance of binary classifiers. Out of all the available metrics, it is acknowledged [21] that Accuracy should not be used to evaluate classifiers when datasets are unbalanced, or rather when there are many normal data points and only a few anomalies. For example, a classifier that always answers "normal" will result in 99% accuracy when testing a dataset where 99% of data points are normal: this does not reflect how "good" such a classifier is. Conversely, MCC equals 0 in this situation, quantifying the classification performance of this "silly" classifier as random guessing. Furthermore, some compound metrics do not account for all 4 classes of

the confusion matrix. For example, F-Measure and all the F-Scores do not use TN to compute their score, leaving an important group of predictions out of the picture.

This is even more important when looking at multi-class classification, which has many classes that may be more or less likely depending on the problem. In addition to MCC, multi-class classifiers are often evaluated through balanced accuracy [17], which weights scores related to each class with its posterior distribution, making for an overall fair evaluation metric, or using the Area Under the ROC curve (AUC).

2.9 Quantifying Task Complexity

Previous studies conjectured that when data points have different labels but feature values that are not distinguishable, the task becomes very complex and prone to misclassifications. This was quantified [34, 49, 58] using *c-measures*, which were first formulated by Ho and Basu [34]as follows.

- *Measures of overlaps in the feature values from different classes.* They measure how features separate examples of different classes. Measures include the maximum Fisher's discriminant ratio (F_1), the overlap of the per-class bounding boxes (F_2), the maximum (individual) feature efficiency (F_3), the directional-vector maximum Fisher's discriminant ratio (F_1v), and the collective feature efficiency (F_4).
- *Measures of class separability.* They estimate to what extent the classes are separable by examining the class boundary. This translates into the minimized sum of the error distance of a linear classifier (L_1), the training error of a linear classifier (L_2), the fraction of points on the class boundary (N_1), the ratio of average intra/inter class nearest neighbor distance (N_2), and the leave-1-out error rate of the 1-nearest neighbor (N_3).
- *Measures of geometry, topology, and density of manifolds.* They provide an indirect characterization of class separability as nonlinearity of i) a linear classifier (L_3) and ii) one-nearest neighbor classifier (N_4), the fraction of maximum covering spheres (T_1), and average points per dimension (T_2).

C-measures are very useful in determining if and how class boundaries are well-separated, and thus they can be used to select features [49]. However, they cannot be used straightforwardly to predict classification performance. This was motivated in [58], where authors attempted to identify correlations of c-measures against Accuracy and Area Under ROC curve (AUC) that resulted from the application of 4 classifiers on different microarray datasets. They conclude that c-measures and resulting accuracy/AUC values are only loosely coupled and cannot be used to precisely estimate metric values.

There are indeed a couple of works that quantify the complexity of a dataset using means other than c-measures. In [50], authors propose complexity descriptors to explain the geometrical distributions of classes in the feature space and the advantages of adopting artificial data sets synthesized according to the distribution of classes. Instead, the work [61] shows how the classification error of ensembles of k-NN classifiers is linked to the complexity of a dataset, refining the N_2 and N_3 c-measures. Unfortunately, neither [61] nor [50] draft strategies to predict the classification performance of classifiers or hint at mechanisms to use those complexity measures for means other than comparing the structure of different datasets.

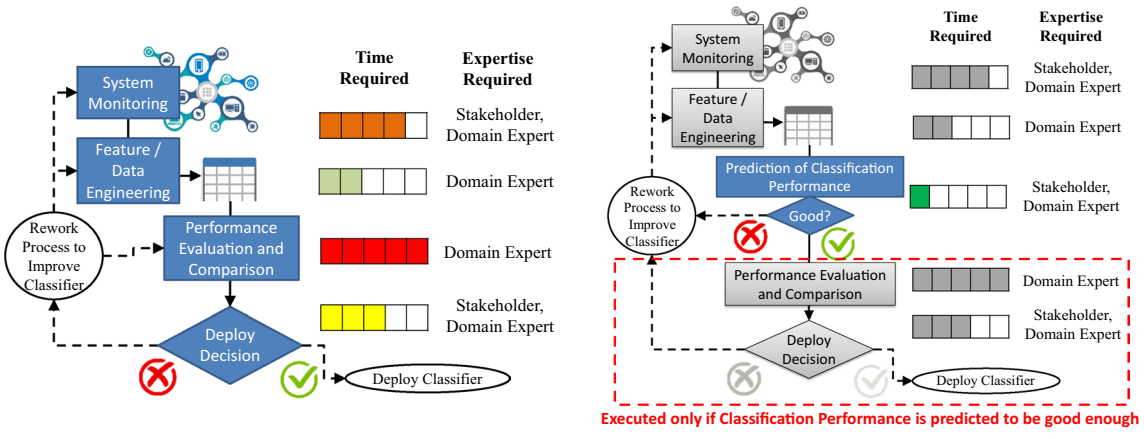

Fig. 1 Workflow for deploying an classifier for a system. The typical process, from [6], is on the left (Fig. 1a), while the process we propose, integrates a strategy to predict classification performance, is on the right (Fig. 1b)

Overall, there is a strong need to investigate different approaches to predict classification performance and quantify metric values to be achieved by classifiers.

3 A Workflow for Predicting Classification Performance

3.1 The Process for Deploying Classifiers

The process of developing a classifier for conducting a specific task in a given system typically follows a workflow composed of different steps [6]. Figure 1a shows a workflow that stems from research papers but also adds information about the timing required to complete each step alongside the expertise required by the system designer. The workflow is composed of 4 main steps we summarize below.

System Monitoring. The design and development of learning-based systems usually starts with the acquisition of a representative dataset. This has to be obtained by the stakeholder – system owner – by monitoring the target system during normal operations over a period of time, collecting performance indicators such as resource usage, system-level or application-level activity. When planning monitoring activities, the stakeholder and the data analyst, or domain expert, should carefully choose the relevant areas, software, hardware or interfaces to monitor, to maximise the information content that is being gathered by monitors during operation. Often, the system gets stress tested while being monitored, to log how performance indicators react to anomalies: this provides information that will be extremely useful when training classifiers. The monitoring strategy is usually set up by the stakeholder (who owns the system and most likely already monitors some key components of the system) alongside the domain expert (e.g., ML expert, data analyst). Both are also responsible for labelling monitored data in case labels will be needed for training classifiers.

Feature / Data Engineering. Collected data needs to be structured, pre-processed, and normalized whenever needed. This step is critical as most of the datasets had multiple data points with missing or mistyped values, or even some observations using different units of measure within the same feature. The pre-processing activity also aims at removing those features that are constant, do not carry information content (e.g., duplicate features with different names, or features that are a simple linear combination of existing features), and removing duplicate label columns, which may negatively affect the overall analysis. Textual or categorical features – if any—are analysed individually to understand if they should be discarded (e.g., the ID/Code of a device), or transformed using strategies as one-hot encoding or entity embedding [71]. This process could also create further meta-features to be provided to classifiers alongside existing features coming from monitoring activities i.e., each monitored performance indicator provides at least a feature for detection. r

Performance Evaluation and Comparison. According to the "free lunch theorem", there exists no universal learning algorithm that outperforms all other approaches in general [85]. As such, it is vital to compare as many classifiers as possible and choose the one that shows the best performance on a specific test set. In this step, the domain expert has to conduct massive work for selecting classifiers, discovering optimal values for hyper-parameters, training, testing, and comparing the metric scores they achieve.

Deploy Decision. The results of the previous steps are used to decide if the system will benefit from the introduction of the classifier or if the process needs to be reworked to be useful in practice. The potential improvement in performance is quantified by the domain expert, which sends their proposal to the stakeholder, who, in turn, knows the requirements that shall be met per applicable standards and deploys the final decision. For example, standards such as the IEC61508 [11, 80] define that the *probability of failure on demand* (Braband, Vom Hövel and Schäbe, 2009) of some components should not exceed a given threshold. Should the classifier fail (e.g., false positives, false negatives, or misclassifications in general) too frequently, we may need to rework (left of Fig. 1) the whole process that involves all the previous steps of the workflow, potentially needing to start again from scratch.

3.2 On Predicting Classification Performance

Failing to deploy a classifier due to poor classification performance has a detrimental impact on the whole system engineering process. In this case, both the stakeholder and the domain expert have wasted a lot of time, resources, and thus money to craft a classifier that never had the potential to be deployed in a real system. It would have been better to suspect such a decision in advance: this way, the stakeholder and the domain expert may have been focusing more on monitoring relevant features or on a more sophisticated feature engineering process rather than wasting time in performance evaluation and comparisons, which is the most demanding step of the whole workflow in Fig. 1a.

Figure 1b shows how a strategy to predict classification performance could interact with the typical workflow. This prediction strategy necessarily needs to be fast to execute and should quantify the performance of classifiers that will be trained using data obtained from monitoring the system and after the feature engineering step. Performance Evaluations and Comparisons will be conducted only if the prediction satisfies the requirements set by the stakeholder, or by standards applicable in the domain. Otherwise, the

stakeholder and the domain expert will focus on improving data rather than exercising classifiers, saving key time.

4 Feature Rankers to Predict Classification Performance

In this section, we present and formalize our strategy to predict classification performance.

4.1 Formal Definition of the Prediction Strategy

We provide a more detailed description of the prediction strategy below. We define:

- F, a set containing k dataset features and their values for each of the data points in the dataset,
- $FR = \{FR_i,\ 1 \le i \le n\}$. a set of n feature rankers,
- *met*. the classification metric to be predicted for a dataset,
- *type* ϵ {*multi*, *sup-bin*, *uns-bin*}, the type of classifier (either supervised or unsupervised, binary of multi-class) we want for a given task.

F, FR, *met*, and *type* are inputs to the prediction of the classification performance and are shown on the left of Fig. 2a. From top to bottom in the same figure, we observe the following.

Each feature ranker FR_i calculates the rank for each of the k features in F. This creates a k-tuple r_i as follows.

$$r_i = FR_i(F) = \left\{value\_f = FR_i(f), f \in F\right\}, 1 \le i \le n, \quad with |r_i| = |F| = k \tag{1}$$

Having n feature rankers FR, a total of n k-tuples are produced as in Eq. (1). However, our prediction strategy should apply to any tabular dataset: thus, we need to find a way to normalize feature rankings into a set of m items regardless of the amount k of features contained in the dataset. In any other case, there will be no way to have a unique predictor of a metric value for any tabular dataset, as the number of rankings assigned by feature rankers will vary a lot. In other words, we need one or more normalization steps $NORM = \{NORM_z,\ 1 \le z \le ns\}$, where each normalization step $NORM_z$ reworks each r_i into a normalized score $NS_{iz} \in \mathbb{R}$ as in Eq. (2).

$$NS_{iz} = NORM_z(r_i) = NORM_z(FR_i(F)), 1 \le i \le n, 1 \le z \le ns \tag{2}$$

This normalization step has to be planned carefully to avoid loss of information compared to using feature ranks r_i as they are provided by feature rankers. First, we take rankings r_i and sort them from the most relevant to the least relevant rank. Sorted ranks can then be aggregated into a wide variety of normalized scores: examples include, but are not limited to: best rank, average of the best *3/5/10* ranks, and sum of all ranks. The union of normalized scores NS_{iz} for each i and z builds *Feature Data*

$$FD = \left\{NS_{iz}, 1 \le i \le n, 1 \le z \le ns\right\}, |FD| = n \cdot ns \tag{3}$$

FD will constitute the input to a regressor *Reg_met@type* that will output a continuous number

$$pred\_met@type = Reg\_met@type(FD) \tag{5}$$

which quantifies the predicted value of the metric *met* for classifiers for a classification problem of a given *type*. Many regressors can be crafted depending on the *met* and *type* in

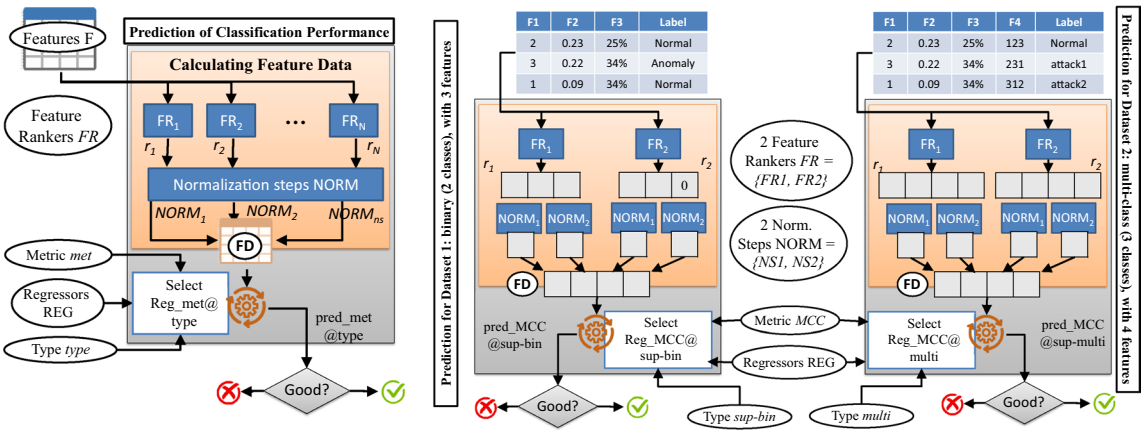

Fig. 2 Detail of our strategy to predict classification performance. On the left (Fig. 2a) we depict an high-level view of the strategy, while on the right(Fig. 2b) we show two examples using datasets with different structure and different problem (binary and multi-class classification), targeting MCC as metric to predict

Eq. (4), building the REG set of regressors. Details on the regressors and their instantiation are in the next section.

Let us clarify this formal definition through the examples in Fig. 2b. The figure shows the flow of data from the beginning to the end of the prediction strategy using two datasets: one with 3 features and 2 class labels, another with 4 features and 3 class labels. In both cases, features are processed by two generic feature rankers FR1 and FR2, thus $n=2$. Each delivers its rankings for each dataset feature: thus, the rankers will deliver 3 values each for the first dataset on the left of Fig. 2b, and 4 values each for the other dataset. Now, we employ two normalization steps NORM_1 "best ranking" and NORM_2 "sum of all rankings", $ns=2$. Regardless of the number of dataset features, NORM_1 and NORM_2 will output a single value for each feature ranker, creating a FD of $n*ns=2*2=4$ items for both datasets. Then, we have to choose the regressor that knows how to predict a specific metric for a specific task. Here, we target the MCC metric and want to predict the metric value for supervised classifiers: these will be binary classifiers for the first dataset, which has two classes (normal, anomaly), and multi-class for the other dataset, which has 3 classes (normal, attack1, attack2).

4.2 Regressors to Predict Classification Performance

Regression is defined as [23] "*a set of statistical processes for estimating the relationships between an outcome variable and one or more independent variables*" (i.e., features). Typically, regression models (regressors) are implemented as supervised ML algorithms (Behera et al., 2023) that predict a numeric ordinal label rather than a class. This perfectly fits the prediction of metric values, which are always expressed as a numeric value, usually in the range [0; 1].

Our study aims at predicting the value of a given metric *met* that will be achieved by the best classifier to perform a given dataset or system: this is the classifier that will be deployed if requirements are met. Therefore, we expect regressors to predict an estimation of the value achieved by the best classifier for a given *task* and a metric *met*. Training regressors require a dataset composed of the FD for many datasets and the associated metric value that has to be computed and works as the ground truth. This requires running several ML algorithms, computing metric values, and comparing them to choose the one that achieved the best value of a given metric in each dataset. Whereas this process is tedious, time-consuming, and requires domain-specific expertise, it is required only for generating training data: once the regressors are trained, they only need Feature Data (classifier-agnostic) to predict classification performance for any tabular dataset.

4.3 Observations

We conclude this formal definition by pointing out the following observations.

- The size of FD does not depend on the number k of features contained in F thanks to the normalization step. Instead, FD always contains $n*ns$ values to be provided to the regressors, with n being the number of feature rankers and ns the number of normalization steps. Once trained, regressors can be applied to datasets with different amounts of features without requiring any additional tuning.
- The outputs of a regressor for a given *task* are independent of the classifier to be used at a later stage and represent the highest expected metric value for a given problem. This simulates the deployment of the best classifier out of a set of candidate classifiers that are exercised and compared according to metric values.
- Feature selection and ranking is of utmost importance in any feature / data engineering process and are computed by design in many data analysis processes. The prediction strategy partially exploits this existing information, minimizing its overhead compared to the usual analysis process.

5 Experimental Campaign

This section describes the experimental campaign to craft our strategy to predict classification performance, paving the way for discussions in the next section. The section develops as follows:

- Sect. 5.1 describes the datasets we used in our experimental study, and details our process for creating variants of such datasets for the purpose of data augmentation.
- Sect. 5.2 and Sect. 5.3 describe how we implemented the feature ranking FR and normalization steps NORM to generate Feature Data.
- Sects. 5.4 to Sect. 5.6 elaborate on how we trained regressors REG to predict classification performance. Section 5.4 and Sect. 5.5 show how to generate labels for training the ML algorithms we use as regressors we list in Sect. 5.6.
- Finally, Sect. 5.7 describes the methodology to conduct experiments, and the machinery we used to implement it.

Table 1 Name, release year, number of attack types, number of portions, and the amount of 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
	[8]	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

5.1 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.

Importantly, datasets always have a "normal" class and at least another class. This allows conducting binary classification even in datasets with multiple labels: in this case, the label is converted into a binary label separating the normal against all other classes (seen as a unique "anomaly" class).

5.1.1 Network Intrusion Detection (NIDS)

We selected labelled datasets on network intrusions looking in surveys [69], *Kaggle, UCI, Zenodo, IEEEDataport* and other online portals. Our selection process resulted in the following datasets: ADFANet [68], AndMal17 [41], BAIoT Doorbell [54], CICIDS17 [33], CICIDS18 [33], CIDDS (Sharafaldin, Habibi Lashkari and Ghorbani, 2018), IoT Network (Kang et al., 2019), ISCX12 [78], NSLKDD [82], UGR16 [51], UNSW-NB15 [59]. 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).

5.1.2 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 [8] 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 (Baidu Inc, no date) competition whose input datasets are still available for download.

5.1.3 Error/Anomaly Detection

The last group of datasets we consider comes from IoT or ICS systems: distributed control systems of a power plant controlling a turbine [76],Shin Hyeok-Ki; Lee and Min, 2023), malfunctions of metros in Portugal [22], railroad trucks equipped with sensors to monitor brake pressure ('APS Failure at Scania Trucks', 2017), an edge device monitored for errors (Zoppi et al., 23AD), the mechanical failure of electrical machinery in power plants [1], and a simulated multivariate time series acquired by sensors on a SpectraQuest's Machinery Fault Simulator [53].

5.1.4 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 into CSV format using *tshark*. Then, we remove features that are specific to the setup that was followed to gather data, namely: Timestamp, ID, and 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. Lastly, we zero-filled all blank values in the *BackBlaze* datasets.

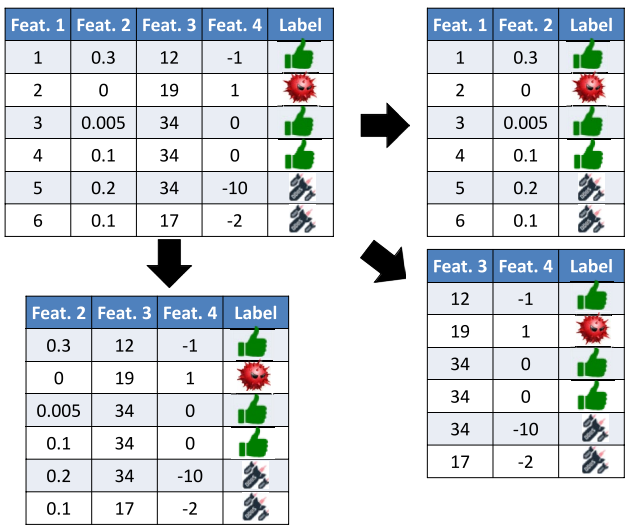

Fig. 3 Data Augmentation used in this work: partitioning features and creating datasets variants

5.1.5 Data Augmentation: Variants of Datasets

We gathered as many datasets as possible to provide a solid baseline to set up and train our strategy to predict classification performance. Unfortunately, even after such an effort, we only have 23 datasets from which we can calculate feature rankings and use them to train and test our predictor. Therefore, we figured out a way to create variants of each dataset (depicted in Fig. 3), which we will be considering as additional datasets.

Particularly, we created variants of each dataset that contain only a subset of features plus the label. As can be seen in Fig. 3, the resulting variants contain the same amount of data points, but only a portion of the features of the initial dataset. This allows the creation of many variants, especially if the initial dataset has many features. Noticeably, creating variants containing only a few features may create variants where data points contain too little information to be considered relevant for our analysis. We experimentally found our sweet spot in creating variants that contain at least 3 features. This means that datasets that have less than 4 features (i.e., ADFANet) will not be used to create variants.

This way, we created 557 variants that, alongside the 23 initial datasets, led the overall number of datasets to be used to train and test our prediction strategy to 580.

5.2 Feature Rankers and FRAPPE

We identify 12 feature rankers based on literature reviews. Unfortunately, there is no available framework that allows computing scores of such a wide variety of feature rankers on the same dataset according to a unified methodology. Therefore we created FRAPPE (Anonymous, no date), a Python library that exercises Feature RAnkers to Predict the classification PerformancE of classifiers. FRAPPE wraps feature rankers from many Python packages, mainly Scikit-Learn, SciPy, and SKRebate, which are being invoked with the same input data and are used to collect metric scores according to a rigorous interface. We introduce each of these feature rankers, grouping them into statistical (SR), Relief-based (RR), or wrapper-based (WR) rankers, and summarize their computational complexity with insights on their implementation.

5.2.1 Statistical-Based Rankers (SR)

SR1. *R-Squared* correlation quantifies the linear correlation between two arrays or data series. It measures the proportion of variation in the dependent variable that can be attributed to the independent variable [55] with a score ranging from 0 to 1 (max correlation).

SR2. *Cosine Similarity* [86] considers two arrays as vectors in an inner product space and computes the cosine of

the angle between those two arrays, which therefore ranges from -1 to 1.

SR3. *Spearman Rank* [89] assesses how well the bond between two variables can be described using a monotonic function. A perfect Spearman correlation of + 1 or -1 occurs when each of the variables is a perfect monotone function of the other.

SR4. *Chi-Square* [20]: performs a statistical test that aims at verifying an independence hypothesis between two arrays. Typically, this test has a boolean outcome, but there are also implementations that quantify the degree of independence and that therefore can be considered as a rank.

SR5. *Pearson Correlation* [88] calculates Pearson correlation between two arrays as the ratio between the covariance of two arrays and the product of their standard deviations,in other words, it is a normalized measurement of covariance in the range [-1; 1].

SR6. *Information Gain* [91] stems from the Kullback–Leibler divergence and quantifies the amount of information gained about a variable from observing another variable. It measures the decrease in entropy when the feature is given with respect to when it is discarded.

SR7. *ANOVA* [64]: the ANalysis Of VAriance (ANOVA) is used to analyze differences in means between groups. Particularly, ANOVA is designed for situations where at least one of the two arrays does not comply with a normal distribution.

5.2.2 Relief-Based Rankers (RR)

RR1. *Relief* [84] is known since a long time as an accurate yet computationally and memory-expensive algorithm to calculate feature ranking. Original Relief was limited to only two-class problems, but has since been extended to multi-class problems and originated a wide variety of alternative implementations.

RR2. *SURF* (Spatially Uniform ReliefF [31]) is an extension of Relief that is more effective in quantifying the correlation of features in noisy datasets.

RR3. *MultiSURF* [84] further extends SURF, performing better than Relief for identifying pure dependencies between features, and yields the most reliable feature selection performance across a wide range of problem types.

RR rankers are known to be very slow; they cannot process a whole dataset in a reasonable time. Consequently, we calculate them on a maximum of 10 000 data points for each dataset and variant. This motivates the * matched to the RR1, RR2, and RR3 rankers in 2: computational complexity in the table uses a variable *ds* as the size of the dataset, but this never exceeds $ds = 10\,000$ in our experiments when using RR Relief-based rankers.

5.2.3 Wrapper-Based Rankers (WR)

Wrapper-based rankings rely on an external classifier which is trained for the sole purpose of deriving feature importance for building their model. Almost any classifier can be used as a wrapper-based ranker. For our study, we wrap two different classifiers: the tree-based *Random Forests* (WR1, [16]),

Table 2 Summary and Computational Complexity of Feature Rankers, where *ds* = "data points (row) in the tabular dataset", *f* = "number of features" = |F|

Tag	Ranker Name	Compl. O()	Comment to the complexity analysis
SR1	R-Squared	$f^2(ds+f)$	Uses minimum least squares for linear regression and extract R^2
SR2	Cosine	$ds \cdot f$	Performs an array to array multiplication, repeated for each feature
SR3	Spearman	$ds \cdot f$	Computes two averages and 2*ds* differences for each feature
SR4	Chi Squared	$ds \cdot f$	SKLearn implementation performs Chi-squared in O(ds·f)
SR5	Pearson	$ds \cdot f$	Array to array computation, repeated for each feature
SR6	MutualInfo	$f \cdot ds \cdot \log(ds)$	SKLearn implementation uses neighbour-based optimization with k = 3
SR7	ANOVAF	$ds \cdot f$	SKLearn has a 1-way ANOVA F-test we iterate for each feature
RR1*	Relief	$ds\,(f+\log(ds))$	Implementation in skrebate runs constant iterations. It uses a K-D tree to speed-up as much as possible its computational time
RR2*	SURF	$ds^2 \cdot f$	The skrebate package does not provide optimized implementations as for the base version of Relief, thus complexity gets slightly higher
RR3*	MultiSURF	$ds^2 \cdot f$	
WR1	Random Forest	$f \cdot ds \cdot \log(ds)$	Feature Selector uses nt = 100 in the forest, no depth limit on trees which are built using a tenth of the overall number of training data points n
WR2	Lin Regression	$f^2(ds+f)$	Using minimum least squares to compute regression and extract coefficients

and the Linear Regressor (WR2, (Behera et al., 2023)). They follow two completely different learning processes and thus provide very different viewpoints on feature ranking. The computational complexity of those wrapper-based feature rankers in 2 has to be interpreted according to three observations:

- Computational complexity in the table refers to the training phase of wrapped classifiers and regressors, which is heavier than the test phase for all WR.
- Wrapped classifiers and regressors are trained and tested with a 50–50 split, so the n in Table 2 is half the size of the dataset. This does not impact computational complexity but it significantly reduces the actual time needed to compute WR rankers.

5.3 Normalization of Ranks

We employ a set of normalized scores NS of 6 items as follows. The first item s_1 is the score assigned to the most relevant feature in F by a given feature ranker. The second, third, and fourth items s_2, s_3, s_4 are the average scores of the 3, 5, 10 features in F that are more relevant according to a feature ranker. Lastly, we compute s_5 and s_6 as the average and the sum of all ranks. In other words:

NS = {s_1: best feature score, s_2: average of the best 3 feature scores, s_3: average of the best 5 feature scores, s_4: average of the best 10 feature scores, s_5: average of all feature scores, s_6: sum of all feature scores}.

Note that s_2, s_3, s_4 can be calculated even if there are fewer ordinal features (i.e., $|F| < 3 / 5 / 10$) than those needed to calculate averages. In this case, the average is calculated by using all the available features. Normalization of scores is already integrated with FRAPPE (Anonymous, no date) we use to calculate scores of feature rankers.

5.4 Supervised Classifiers

We select different supervised classifiers to analyze each dataset and variant for creating labels to train regressors. We are interested in selecting a subset of classifiers that are as heterogeneous as possible to avoid exercising many classifiers which will result in very similar outcomes. We favor classifiers that require minimal parameter tuning to avoid conducting random or grid searches which would add yet another dimension of analysis, and make sure to include those classifiers that are known to be very good at classifying tabular data [16, 30, 79], plus other alternatives. As discussed in Sect. 2.3, we disregard using neural networks as they are not recommended for classifying tabular data.

Therefore, we selected the statistical Naïve Bayes Linear Discriminant Analysis and Logistic Regression [36], as well as the tree-based Extra-Trees, LogitBoost, and Extreme Gradient Boosting [24],Geurts, Ernst and Wehenkel, 2006; [19] whose implementations are all made available in the *Scikit-Learn*, *logitboost* and *xgboost* Python packages. We did not consider slow classifiers like Support Vector Machines and K-th Nearest Neighbors (even with the kd-tree enhanced neighbor search) as the time needed to complete experiments was already requiring weeks.

5.5 Unsupervised Classifiers

Then, we need to generate labels for training regressors that predict the classification performance of unsupervised classifiers. To do so, we select a set of unsupervised classifiers that are as heterogeneous as possible and span across different families of unsupervised classifiers; we ended up selecting one algorithm for each family in [29, 94], namely *G-Means* (clustering family), *HBOS* (statistical), *SOM* (neural-network), *Isolation Forests* (iForest, classification), *ODIN* (neighbour-based), *SDO* (density-based), *FastABOD* (angle-based), autoencoders (neural networks). We exercise those unsupervised classifiers by using the library PYOD (Zhao, Nasrullah and Li, 2019).

5.6 Regressors REG

Once feature data and classification metric values have been calculated, we can train the regressors REG. Particularly, we are interested in classifying binary and multi-class datasets, with supervised and unsupervised classifiers. Thus, we will have *type*$\in${*sup-bin, uns-bin, multi*}. Also, we will be targeting the prediction of metrics that are robust to unbalanced datasets and fit binary and multi-class classification. From the discussion in Sect. 2.5, we choose *met*$\in${*mcc, auc*}. The 6 combinations of *type* and *met* values result in a total of 6 regressors to be trained:

REG = {Reg_mcc@sup-bin, Reg_auc@sup-bin,
Reg_mcc@uns-bin, Reg_auc@uns-bin,
Reg_mcc@multi, Reg_auc@multi}.

Each of these 6 regressors can be implemented as one of the supervised ML algorithms available in the literature that are capable of predicting a numeric label. To select the best ML algorithm for each of the 6 REG regressors, we train and compare several supervised regressors with heterogeneous characteristics:

- statistical algorithms [90], Behera et al., 2023) as Linear Regression, Lasso Regression, Decision Trees,
- bagging [16], Geurts, Ernst and Wehenkel, 2006) meta-learners (Random Forest, Extremely Randomized Trees) and

- boosting [19] meta-learners (XGBoost).

We apply each of those 6 ML algorithms for each REG regressor and we compute the Mean Absolute Error (MAE) and R-Squared correlation to measure their goodness of approximation to the numeric label. The algorithm with the lowest MAE will be chosen to implement each regressor REG and will be used to predict classification performance.

We separated the train and test partition making sure that datasets or variants were either in the training or in the test set to avoid "contamination". Overall, 17 datasets and their variants were used as train set, letting the other 6 datasets (with their variants) build the test set for a rough 70–30% split. The 6 datasets in the test set were chosen as follows: 2 intrusion detection, 2 hardware failure, and 2 error detection datasets. Then, we cross-validated the train-test process by changing the datasets used for training and testing, noticing only negligible changes. Prediction results presented in the next section use one of the models created within the cross-validation process, chosen randomly during the process.

5.7 Methodology to Conduct Experiments

We downloaded the datasets from their repositories and extracted the variants as described in Sect. 5.1. Then, we processed all the resulting datasets and variants with the FRAPPE framework (Anonymous, no date), which calculates the ranks of each feature in datasets and variants according to the feature rankers in Sect. 5.2; those are 7 statistical filter-based, 3 Relief filter-based, and 2 wrapper-based rankers. Then, FRAPPE calculates 6 normalized scores (see Sect. 5.3) for each feature ranker, generating Feature Data containing 78 items (13 feature rankers * 6 normalized scores) for each of the 580 datasets or variants. This constitutes the input of the strategy to predict classification performance, which also requires labels for training the regressors *REG*. We exercise supervised and unsupervised classifiers in Sect. 5.4 and Sect. 5.5 on each dataset or variant using a 50–50 train-test split and calculating MCC and AUC metric scores. Collecting these metrics scores allows training the REG regressors (see Sect. 5.6) that will predict classification performance. For completeness of our analysis, we will also repeat the training of regressors using the SMOGN (Branco, Torgo and Ribeiro, 2017) framework for data augmentation through Synthetic Minority Over-Sampling of the set used for training regressors. This allowed doubling the size of the training set: however, this introduces synthetic data that may alter the behavior of regressors and as such it is discussed in the next section.

Experiments have been executed on an Intel Core i7-6700 with four 3.40GHz cores, 24GB of RAM, and 1TB of storage, and required approximately three weeks of 24H execution. All the scores and files we used in the paper are available at (Anonymous, no date), folder "scripts".

6 Discussion and Implementation of the Prediction Strategy

Here we present and discuss the prediction strategy that results from the application of the experimental campaign in Sect. 5. Section 6.1 elaborates on how well our strategy predicts MCC and AUC scores of supervised and unsupervised classifiers, whereas Sect. 6.2 discusses on feature rankers and how they contribute to building the prediction strategy. Section 6.3 discusses optimizations, while Sect. 6.4 provides insights on the implementation and practical usage of our findings.

Table 3 Mean Absolute Error (MAE) and R-Squared for the best regressor to predict either MCC or AUC for supervised (Sup-bin, multi) and unsupervised (Uns-bin) classifiers

Regressor	ML Algorithm	Type*type*	Metric*met*	SMOGN Usage	MAE	R-Sq
Reg_MCC@Sup	XGBoost	sup-bin	MCC	No	0.069	0.913
				Yes	0.089	0.817
Reg_MCC@Unsup	XGBoost	uns-bin		No	0.085	0.854
				Yes	0.096	0.825
Reg_MCC@Multi	ExtraTrees	Multi		No	0.071	0.905
				Yes	Failed	
Reg_AUC@Sup	Random Forest	sup-bin	AUC	No	0.051	0.890
				Yes	0.067	0.782
Reg_AUC@Unsup	XGBoost	uns-bin		No	0.053	0.870
				Yes	0.060	0.816
Reg_AUC@Multi	XGBoost	Multi		No	0.050	0.892
				Yes	0.060	0.808

6.1 Evaluation of Regressors using all Feature Rankers

We explore the prediction error of our strategy with the aid of Table 3, which reports the Mean Absolute Error (MAE) and R-Squared correlation for each of the 6 REG regressors. For completeness of analysis, we report also scores in which we trained regressors using the SMOGN data augmentation: this increases the size of the training set, but has no impact on the test set. The table immediately suggests three discussion items.

- MAE scores are higher for regressors predicting MCC (i.e., Reg_MCC@Sup-bin, Reg_MCC@Uns-bin, Reg_MCC@Multi in the first half of the table) compared to regressors predicting AUC (bottom of the table). This may seem to hint at a poor approximation of MCC compared to AUC: however, the reader should note that AUC scores mostly fall in the range [0.5 – 1], whereas MCC usually falls in the [0 – 1] interval. Therefore, having a higher MAE for MCC regressors does not necessarily mean that they are not as effective as regressors that predict AUC values. This explanation is supported by the R-Squared correlation scores, which hover in the range of 80—90 for all regressors.
- Applying SMOGN augmentation for training regressors did not have a beneficial impact on the regression task itself. All rows of Table 3 with a "Yes" in the fourth column show worse scores compared to their counterparts with "No" data augmentation. This may

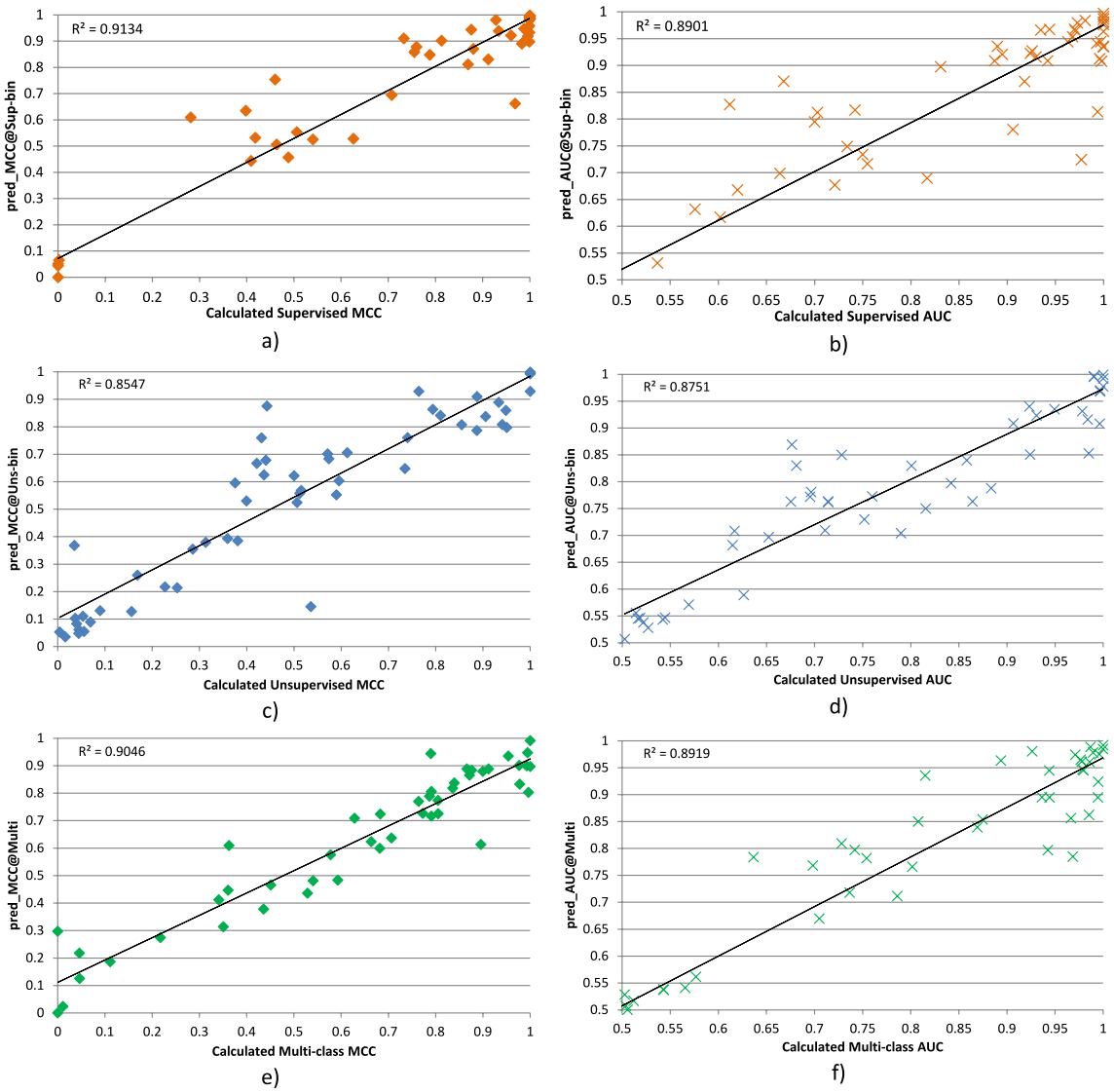

Fig. 4 Scatterplots showing calculated against predicted metric scores for the 6 regressors REG. From top-left to bottom right: 4a) Reg_MCC@Sup-bin, 4b) Reg_AUC@Sup-bin, 4c) Reg_MCC@Uns-bin, 4d) Reg_AUC@Uns-bin, 4e) Reg_MCC@Multi, 4f) Reg_AUC@Multi

be due to a multitude of reasons: for instance, the SMOGN algorithm did not have enough data to learn how to generate novel data belonging to the initial distribution, with a detrimental effect on the learning phase of regressors. In one case the library even failed regardless of all the tries we made (see Reg_MCC@Multi in Table 3)

- For 4 out of 6 regressors, the XGBoost regressor was the one delivering the lowest MAE and thus chosen as the best model. From a more general standpoint, among those exercised as candidate regressors, we observed how Linear Regression, Lasso Regression, and Decision Trees ended up having a noticeably higher MAE (worse predictive capabilities) than XGBoost, Random Forest, and ExtraTrees.

We further explore the results through the scatterplots in 4. Each of the 6 scatterplots depicts the calculated metric score against the predicted value of a regressor in REG; Fig. 4a and Fig. 4b show the behavior of regressors predicting MCC (the former) and AUC (the latter) scores of supervised binary classifiers, Fig. 4c and Fig. 4d are related to regressors predicting classification performance of unsupervised binary classifiers, whereas Fig. 4e and Fig. 4f are related to regressors predicting classification performance of multi-class (supervised) classifiers. The diagonal black line in each scatterplot shows the linear approximation used to calculate the R-Squared value.

The plot in the top-left of the figure depicts the graphical representation of the first line of Table 3. Diamonds in the plot are mostly in the upper-right corner: those correspond to datasets or variants where the best supervised binary classifier got a very high MCC score, and at the same time they were predicted to have a high MCC. On the contrary, orange diamonds in the bottom-left corner of the plot point to datasets or variants where no supervised classifier was able to achieve a high MCC and that were also predicted to have a poor MCC. Noticeably, there are diamonds in the plot that fall far away from the black line. Those below the line correspond to datasets or variants where the predicted pred_MCC@Sup-bin value was inferior to the calculated MCC: in this case, our prediction strategy overestimated the difficulty of the dataset. Diamonds above the black line are instead those in which the Reg_MCC@Sup-bin regressor predicted a value that is a lot higher than the calculated MCC, underestimating the classification performance in the dataset or variant.

The other plots in Fig. 4 have a trend similar to the plot we examined before. Another interesting observation is that blue diamonds and crosses in Fig. 4c and Fig. 4d are overall closer to the bottom left compared with the first two plots. This is because supervised classifiers (first two plots) output a lower amount of misclassifications than unsupervised classifiers (two plots in the middle), causing their metric scores to be higher and overall closer to the top right of plots. The trend for multi-class classifiers in Fig. 4e and Fig. 4f follows that of the plots above, with no major changes.

Table 4 Correlation of normalized scores of feature rankers with metric to predict (score correlation), importance as features when training regressors, and average time needed to calculate rankings

Ranker Tag	Ranker Name	Score Correlation	Feature Importance	Time (s)
SR1	R-Squared	0.142	0.004	0.081
SR2	Cosine	0.011	0.005	0.044
SR3	Spearman	0.045	0.008	0.678
SR4	Chi Squared	0.029	0.004	0.735
SR5	Pearson	0.046	0.003	0.142
SR6	MutualInfo	0.173	0.092	18.591
SR7	ANOVAF	0.009	0.011	0.196
RR1	Relief	0.143	0.011	12.652*
RR2	SURF	0.019	0.006	47.443*
RR3	MultiSURF	0.019	0.013	30.797*
WR1	RandomForest	0.041	0.021	0.318
WR2	Linear Regression	0.006	0.004	0.224

6.2 Contribution of Feature Rankers

Another important discussion item is related to whether each of the 12 feature rankers in this study contributes to predicting classification performance. Particularly, we aim to understand if there are feature rankers that carry little to no contribution to the prediction strategy and therefore could be dropped to speed up the process. Table 4 shows two types of relevance measures for each feature ranker:

- *Score Correlation*, which averages the R-correlation between normalized scores of a ranker with the metric value to be predicted,
- *Feature Importance*, or rather the average of relevance scores assigned by regressors to each feature at the end of the training process, and
- *Time (s)*, the average time (seconds) needed from the feature ranker to process a dataset or variant and compute ranks. This varies with the size of the dataset but still gives an actual indication of the complexity of each ranker.

We immediately observe that SR1, SR6, RR1, and, to a lesser extent, WR1, have higher scores for the first two measures than other feature rankers. Feature Importance

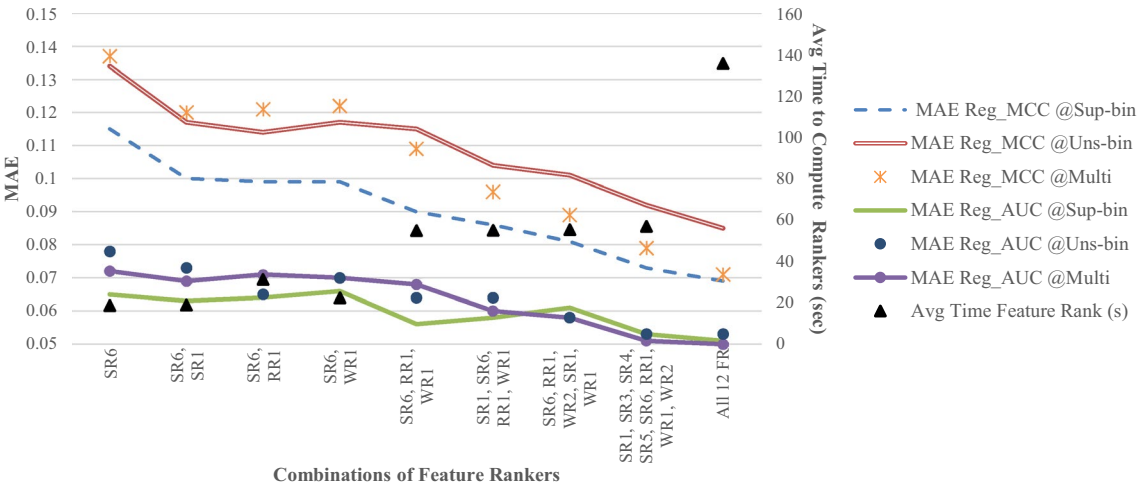

Fig. 5 Variation of MAE for the 6 REG regressors and time (sec) for different combinations of feature rankers

(4th column) highlights that SR6 has a prominent role when training regressors, whereas other rankers have only a marginal contribution. Regarding the time needed to compute scores of feature rankers, it is clear that SR6, all RR, and WR2 are far slower than other feature rankers. Moreover, in our experiments, we ran RR rankers using only a small portion of datasets and variants: that is why we starred (*) those timings in the table. The normalization process has no impact on those relevance measures: normalizing scores takes only a few microseconds of execution time.

6.3 Optimizing the Prediction Strategy

This analysis suggests the possibility of refining the prediction strategy, improving its speed, and maintaining similar regression performance. According to Table 4, rankers SR2, SR7, RR2, RR3, and WR2 seem to have a negligible impact on our strategy and therefore are a candidate to be removed from the set of feature rankers. Also, RR2 and RR3 are the most time-consuming rankers and therefore it would be very beneficial to build a predictor that does not need those two rankers when building feature data.

However, dropping feature rankers may impact the goodness of predictions of our strategy: we measure it by iterating training of regressors REG using different subsets of feature rankers, and plot results in Fig. 5: On the x-axis, we plotted different combinations of feature rankers used to generate Feature Data; lines represent the MAE for the 6 regressors in REG, while triangles show the average time needed to compute each combination of feature rankers (plotted on the secondary y-axis, see on the right of the plot in the figure). We can observe an almost constant decrease in the MAE from left to right, corresponding to using more and more feature rankers, up to a point in which all rankers are used (at the extreme right of the plot, MAEs match those in Table 3). The second last combination contains all feature rankers but those that seemed to have negligible impact on the overall process i.e., SR2, SR7, RR2, RR3, and WR3. These MAEs are slightly higher than those obtained using all feature rankers, meaning that even the rankers that seemed to contribute little to the prediction strategy have a role in predicting metric scores. On the other hand, this has a clear advantage as it more than halves (57 instead of 136) the time needed to compute feature rankings and thus predict classification performance.

As a result, we may predict misclassifications using only 8 feature rankers (i.e., SR1, SR3, SR4, SR5, SR6, RR1, WR1, WR2). This guarantees a very fast computation time at a cost of a slightly higher MAE. We refer to this group of rankers as OPT, whereas predictors using all rankers are referred to as FULL.

6.4 Implementing the Prediction Strategy in FRAPPE

The regressors discussed in the previous sections are deployed and currently available as part of the FRAPPE GitHub repository at (Anonymous, no date). Users willing to predict the classification performance of tabular data classifiers have to follow the steps in Listing 1.

The *FrappeInstance* object is the main item for the prediction strategy. It has to be initialized depending on the specific task (sup-bin, uns-bin, multi), the metric to be predicted (mcc, auc) and the specific regressors to be used, either OPTimized or using the FULL set of rankers. Then, the dataset has to be loaded using the function provided by the library or custom functions. The only requirement is to load the dataset and *labels* (if any, otherwise *labels* = *None*)

as *numpy ndarrays* or *pandas DataFrames*. At this point, we can call the *pred_met* function. Noticeably, using the prediction strategy is straightforward and allows also non-experts to use it in their research and industrial case studies.

The FRAPPE library offers a wide variety of alternative setups to exercise feature rankers and predict the classification performance of datasets. Also, it allows re-training of regressors for learning to predict other metrics or using different datasets as train baselines. Further information can be found in the repository.

7 Application: Predicting Classification Performance for Different Case Studies

We predict the classification performance of datasets belonging to different domains, even far from anomaly-based detectors to show the generalization capabilities of our strategy. We will be predicting MCC and AUC scores for the following datasets:

- A dataset (https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package) that contains features from environmental monitoring in Australia and aims at predicting is the next day will be a rainy day or not.
- The Titanic survival dataset (https://www.kaggle.com/c/titanic), where features allow us to predict if a passenger survived the accident or not.
- Airplane Satisfaction (https://www.kaggle.com/datasets/teejmahal20/airline-passenger-satisfaction), a dataset that contains information about flights and is used to predict if passengers will be satisfied by the flight or not.
- The hotel booking dataset (Antonio, de Almeida and Nunes, 2019), in which we aim to predict if users will cancel a booking before finalizing the booking to the hotel or B&B.
- The ICU admission dataset (https://www.kaggle.com/datasets/mitishaagarwal/patient), which contains data from patients admitted in US ICUs and is used to learn if a patient will survive or die in the ICU.
- The RT-IoT2022 dataset [75], yet another example of anomaly-based intrusion detection in an IoT environment.

Throughout the process, we measured the time needed to calculate those predictions and log the predicted MCC and AUC scores. During the process, we also exercised classifiers, sensitivity analyses, and connected activities to simulate an effective deployment of these classifiers, measuring the same metric scores and required time. Note that the time that will be required in practice is usually longer than this estimation since the process has to be set up by a domain expert, who should also check results and monitor the process to spot potential errors or misinterpretations of data.

Table 5 Predictions and MAE of our strategy to predict misclassifications on the 6 datasets in this section

Dataset			Task type	True Values			Predictions			Error	
Name	# Points	# Feat		Time (s)	MCC	AUC	Time (s)	MCC	AUC	MCC	AUC
AirplaneSatisfaction	103,904	18	bin-sup	695.2	0.894	0.945	7.9	0.813	0.907	0.081	0.038
			bin-uns	456.6	0.314	0.716	7.8	0.379	0.737	0.065	0.021
			multi	701.9	0.894	0.945	8.0	0.867	0.863	0.017	0.077
HotelBooking	119,390	19	bin-sup	835.4	0.657	0.810	9.0	0.790	0.894	0.132	0.084
			bin-uns	480.2	0.212	0.625	9.1	0.353	0.698	0.141	0.073
			multi	827.1	0.657	0.810	9.0	0.698	0.738	0.076	0.057
ICUSurvival	91,712	74	bin-sup	794.7	0.628	0.667	19.8	0.715	0.760	0.087	0.093
			bin-uns	360.8	0.208	0.688	20.1	0.291	0.719	0.083	0.030
			multi	790.1	0.628	0.667	20.0	0.777	0.890	0.041	0.030
RTIoT22	123,120	81	bin-sup	1342.1	0.989	0.993	30.1	0.992	0.984	0.003	0.009
			bin-uns	1002.2	0.386	0.817	29.6	0.485	0.775	0.099	0.042
			multi	1476.1	0.992	1.000	29.9	0.974	1.005	0.018	0.005
Titanic	890	9	bin-sup	4.3	0.659	0.814	0.4	0.757	0.895	0.099	0.081
			bin-uns	2.7	0.268	0.662	0.3	0.305	0.712	0.037	0.050
			multi	4.4	0.659	0.814	0.4	0.728	0.817	0.105	0.028
WeatherAUS	145,460	21	bin-sup	741.0	0.734	0.767	9.3	0.756	0.848	0.022	0.081
			bin-uns	461.4	0.203	0.626	9.1	0.346	0.694	0.143	0.068
			multi	760.0	0.734	0.767	9.3	0.696	0.781	0.180	0.062
									MAE	0.075	0.050

Results show calculated (True Values in Table 5) and predicted (Predictions in the table) MCC and AUC scores using classifiers for *bin-sup, bin-uns,* and *multi* tasks. For each dataset, the table reports its size, the number of features, the time needed to find the best classifier and the MCC/AUC it gets, the time needed to predict classification performance, and the MCC/AUC it predicts, plus the prediction error of MCC/AUC on the right of each table. True values computed for multi and bin-sup tasks are often the same: this happens when the dataset represents a binary classification problem (all datasets but RT_IoT22). Predicted values are instead different as they use different regressors. We identify two discussion items to explore.

MAE Analysis. First, we want to understand if predictions of classification performance are close to the actual classification performance of classifiers we exercised. In Table 5, this is measured by the last two columns on the right, one for MCC and the other one for AUC. On the bottom right of the table, we computed the average of these prediction errors (MAEs). The MAE for predictions is 0.075 / 0.050 for MCC / AUC, which is similar to the MAEs we presented in Table 3 and discussed in Sect. 6.1.

This is a very important observation as it shows how well the prediction strategy generalizes to any classification problem, even if it is not related to anomaly detection. Out of the 6 datasets in the table, only the RT_IoT22 dataset is closely related to the domain we used to train and validate our predictors, but the goodness of prediction of our strategy is still the same. Overall, our predictors are robust to domain shifts.

Time to Predict. The time needed to exercise classifiers (5th column of Table 5) is at least one – and sometimes two—order of magnitude more than that needed to predict classification performance (see 8th column of the tables). This includes only the "experimentation time", and leaves out all the time needed by the domain expert to plan, monitor, and analyze experiments and their results, which are going to make this difference even more noticeable. This difference is very small when dealing with small datasets (i.e., the Titanic Disaster) but grows a lot the more data points are contained in the dataset. Therefore, we expect the prediction strategy to provide results faster than conducting regular analyses especially when dealing with big datasets, which is a de-facto standard for many applications nowadays.

Interestingly, the ICU Survival dataset has fewer data points than Weather AUS, but the prediction strategy takes more time when processing the ICU dataset. This may seem counter-intuitive: however, the reader should notice that the ICU dataset contains 74 features, while the Weather AUS contains only 21 features. This may suggest that our prediction strategy performs slower when processing datasets with many features and a limited number of data points i.e., microarray datasets. However, those datasets are not as common as those containing far more data points than features: therefore we do not investigate this behavior any further.

8 Concluding Remarks

To conclude the paper, we summarize in this section the conclusions and achievements of our work, limitations to the validity of our study, and future directions.

8.1 Lessons Learned and Achievements

We summarize the findings of this paper as follows.

- Feature rankers allow to predict the classification performance of classifiers when performing supervised or unsupervised classification with low prediction error in a given dataset. This is an important contribution as previous studies using other techniques (i.e., c-measures) seemed to deny this opportunity.
- Prediction errors, measured as MAE are small, but not absent. As such, the prediction of classification performance cannot be trusted in scenarios where even a small change in metric values can make a big difference. Instead, it should be used as a preliminary analysis for evaluating the complexity of a dataset quantified as classification metrics such as MCC or AUC.
- Our strategy to predict classification performance can be integrated into existing data analysis processes requiring little to no modifications to the existing analysis flow, and quantifies the prediction in a few minutes at most. This is due to the FRAPPE library providing user-friendly and easy-to-use support to the analyst, which has to provide a few inputs and can get the predicted classification performance as desired. No expertise is required of the user nor knowledge about feature rankers, as interfaces of the FRAPPE library hide all the implementation details.
- The prediction strategy applies to different tabular datasets, even if not related to the error and intrusion detection domain, predicting the performance of unsupervised binary classifiers and supervised binary and multi-class classifiers with a stable prediction error.

8.2 Limitations of this Study

We report here possible limitations to the validity and the applicability of our study. These are not to be intended as showstoppers when considering the conclusions of this paper. Instead, they should be interpreted as boundaries or possible future implications that may impact the validity of this study.

8.2.1 Usage of Public Data.

The usage of public datasets and open-source tools was a pre-requisite of our analysis as it allows reproducibility and relies on proven-in-use data. However, the heterogeneity of data sources and their potential lack of documentation may limit the understandability of data. In addition, such datasets are not under our control: therefore, possible actions such as changing the way data is generated are out of consideration. For example, we were forced to process datasets that have only a few features e.g., ADFANet (3) as they are.

Additionally, the reader may argue that even our massive effort in processing 23 different public datasets and creating variants resulted only in several hundreds of data points to train, validate, and test the prediction strategy. As already discussed throughout the paper, generating a data point for the regressors requires finding a public dataset, learning its structure, connecting it to FRAPPE, calculating feature data, and exercising supervised and unsupervised classifiers to compute metric scores that will serve as labels for training regressors. We are aware that using only hundreds to a few thousand (using SMOTE data augmentation) data points may open the problem of the robustness of our predictor. However, the tests in Sect. 7 show that the metric predictions for brand-new datasets are within the range estimated during our experimental evaluation.

8.2.2 Parameters of Classifiers.

Each classifier relies on its parameters. Finding the optimal values of parameters is a substantial process that requires sensitive analyses and is directly linked with the scenario in which the classifier is going to be exercised. When applying supervised and unsupervised classifiers to different datasets it is not always possible to precisely tune these parameters. Predicted metric values could turn out to be slightly different from the theoretical optimum due to a more or less optimal tuning of the classifier. The impact on our prediction strategy is considered negligible, as it is obtained on top of extensive experiments using many datasets and many classifiers – ranging from requiring extensive parametrization to none –, smoothing down possible performance degradation due to this event.

8.2.3 Predicting Performance of Regressors.

Our strategy can be adapted with minor modifications to predict the classification performance of regressors. To do that, we may need to reconsider the feature rankers we used in this paper as some of them require a categorical label and may not translate well when the target of the machine learning algorithm is a continuous quantity, as it happens with regressors. However, the general approach still holds.

Acknowledgements No additional acknowledgements aside from the funding already specified above.

Author Contributions First author managed most of the technical work, including data curation, conceptualization, experiments planning and execution. The second author and the third author helped with funding acquisition and in reviewing the paper at different stages. The initial drafts were written by the first author.

Funding 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 funded by the Italian MUR and by the European Union under NextGenerationEU. by the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU, and by the RDS—PTR22-24 P2.1 Cybersecurity project funded within the Ricerca di Sistema Elettrico, Piano Triennale 22–24.

Data availability Data used in the paper is publicly available at the websites of their owners, as referred in the paper (see Sect. 5.1 and Sect. 7). Code developed for this work is available on the public GitHub, which was anonymized at https://github.com/tommyippoz/FRAPPE).

Declarations

Conflict of interests Authors declare no competing interests.

References

1. Agarwal, A. (2018) 'Machine Failure Prediction'. Kaggle. https://kaggle.com/competitions/machine-failure-prediction.
2. Zoppi T (2024) FRAPPE GitHub Repository. https://github.com/tommyippoz/FRAPPE
3. Antonio N, de Almeida A, Nunes L (2019) Hotel booking demand datasets. Data in Brief 22:41–49. https://doi.org/10.1016/j.dib.2018.11.126
4. 'APS Failure at Scania Trucks' (2017).
5. Arik SÖ, Pfister T (2021) TabNet: Attentive Interpretable Tabular Learning. Proc AAAI Conf Artif Intell 35(8):6679–6687. https://doi.org/10.1609/aaai.v35i8.16826
6. Arp, D. *et al.* (2022) 'Dos and Donts of Machine Learning in Computer Security', in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, pp.

3971–3988. https://www.usenix.org/conference/usenixsecurity22/presentation/arp.
7. Avizienis A et al (2004) Basic concepts and taxonomy of dependable and secure computing. IEEE Transact Dependable Secure Comput 1(1):11–33. https://doi.org/10.1109/TDSC.2004.2
8. BackBlaze (2023) *BackBlaze HDD Data*, https://www.backblaze.com/cloud-storage/resources/hard-drive-test-data.
9. Baidu Inc (no date) *Baidu HDD - Baidu SMART Dataset for Seagate ST31000524NS drive model*, https://www.kaggle.com/datasets/drtycoon/hdds-dataset-baidu-inc.
10. Behera J et al (2023) Prediction based mean-value-at-risk portfolio optimization using machine learning regression algorithms for multi-national stock markets. Eng Appl Artif Intell 120:105843. https://doi.org/10.1016/j.engappai.2023.105843
11. Bell, R. (2006) 'Introduction to IEC 61508', in *Proceedings of the 10th Australian workshop on Safety critical systems and software-Volume 55*, pp. 3–12.
12. Bi X, Wang H (2019) An enhanced high-order Boltzmann machine for feature engineering. Eng Appl Artif Intell 78:37–52. https://doi.org/10.1016/j.engappai.2018.10.011
13. Bishop, C. and Nasrabadi, N. (2006) *Pattern Recognition and Machine Learning*. 4th edn. Springer.
14. Braband, J., Vom Hövel, R. and Schäbe, H. (2009) 'Probability of failure on demand–the why and the how', in *Computer Safety, Reliability, and Security: 28th International Conference, SAFECOMP 2009, Hamburg, Germany, September 15–18, 2009. Proceedings 28*, pp. 46–54.
15. Branco, P., Torgo, L. and Ribeiro, R.P. (2017) 'SMOGN: a Pre-processing Approach for Imbalanced Regression', in P.B. Luís Torgo and N. Moniz (eds) *Proceedings of the First International Workshop on Learning with Imbalanced Domains: Theory and Applications*. PMLR (Proceedings of Machine Learning Research), pp. 36–50. https://proceedings.mlr.press/v74/branco17a.html.
16. Breiman L (2001) Random Forests. Machine Learning 45(1):5–32. https://doi.org/10.1023/A:1010933404324
17. Brodersen, K.H. *et al.* (2010) 'The Balanced Accuracy and Its Posterior Distribution', in *2010 20th International Conference on Pattern Recognition*. IEEE, pp. 3121–3124. https://doi.org/10.1109/ICPR.2010.764.
18. Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: a Survey. ACM Comput Surv 41(3):1–58. https://doi.org/10.1145/1541880.1541882
19. Chen, T. and Guestrin, C. (2016) 'XGBoost: A Scalable Tree Boosting System', in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, pp. 785–794. https://doi.org/10.1145/2939672.2939785.
20. Chen, Y. *et al.* (2006) 'Survey and taxonomy of feature selection algorithms in intrusion detection system', in *Information Security and Cryptology: Second SKLOIS Conference, Inscrypt 2006, Beijing, China, November 29-December 1, 2006. Proceedings 2*, pp. 153–167.
21. Chicco D, Jurman G (2020) The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC Genomics 21(1):6. https://doi.org/10.1186/s12864-019-6413-7
22. Davari, N. *et al.* (2021) '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, pp. 1–10. https://doi.org/10.1109/DSAA53316.2021.9564181.
23. Fahrmeir L et al (2013) Regression models. Springer
24. Friedman J, Hastie T, Tibshirani R (2000) Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors). annals stat 28(2):337–407r
25. Garces H, Sbarbaro D (2011) Outliers detection in environmental monitoring databases. Eng Appl Artif Intell 24(2):341–349. https://doi.org/10.1016/j.engappai.2010.10.018
26. Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63(1):3–42. https://doi.org/10.1007/s10994-006-6226-1
27. Gil M et al (2019) Designing human-in-the-loop autonomous Cyber-Physical Systems. Int J Human-Comput Stud 130:21–39. https://doi.org/10.1016/j.ijhcs.2019.04.006
28. De Giovanni E et al (2021) Real-Time Personalized Atrial Fibrillation Prediction on Multi-Core Wearable Sensors. IEEE Transact Emerg Topics Comput 9(4):1654–1666. https://doi.org/10.1109/TETC.2020.3014847
29. Goldstein M, Uchida S (2016) A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data. PLOS ONE 11(4):e0152173. https://doi.org/10.1371/journal.pone.0152173
30. González S et al (2020) A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities. Inf Fusion 64:205–237. https://doi.org/10.1016/j.inffus.2020.07.007
31. Greene CS et al (2009) Spatially uniform relieff (SURF) for computationally-efficient filtering of gene-gene interactions. BioData mining 2:1–9
32. Grinsztajn L, Oyallon E, Varoquaux G (2022) Why do tree-based models still outperform deep learning on typical tabular data? Adv Neural Inf Process Syst 35:507–520
33. Haider W et al (2017) Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. J Netw Comput Appl 87:185–192. https://doi.org/10.1016/j.jnca.2017.03.018
34. Ho TK, Basu M (2002) Complexity measures of supervised classification problems. IEEE Trans Pattern Anal Mach Intell 24(3):289–300
35. Howard J, Gugger S (2020) Fastai: A Layered API for Deep Learning. Inf 11(2):108. https://doi.org/10.3390/info11020108
36. Huang Y, Guan Y (2015) On the linear discriminant analysis for large number of classes. Eng Appl Artif Intell 43:15–26. https://doi.org/10.1016/j.engappai.2015.03.006
37. Kang, H. *et al.* (2019) 'IoT network intrusion dataset'. IEEE Dataport. https://doi.org/10.21227/q70p-q449.
38. Khraisat A et al (2019) Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity 2(1):20. https://doi.org/10.1186/s42400-019-0038-7
39. Krzanowski WJ et al (2006) Confidence in Classification: A Bayesian Approach. J Classif 23(2):199–220. https://doi.org/10.1007/s00357-006-0013-3
40. Kuhn, M. and Johnson, K. (2019) *Feature engineering and selection: A practical approach for predictive models*. Chapman and Hall/CRC.
41. Lashkari, A.H. *et al.* (2018) 'Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification', in *2018 International Carnahan Conference on Security Technology (ICCST)*. IEEE, pp. 1–7. https://doi.org/10.1109/CCST.2018.8585560.
42. Le, L., Patterson, A. and White, M. (2018) 'Supervised autoencoders: Improving generalization performance with unsupervised regularizers', in S. Bengio et al. (eds) *Advances in Neural Information Processing Systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/file/2a38a4a9316c49e5a833517c45d31070-Paper.pdf.
43. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444. https://doi.org/10.1038/nature14539

44. Leroux S, Simoens P (2023) Sparse random neural networks for online anomaly detection on sensor nodes. Future Generation Comput Syst 144:327–343. https://doi.org/10.1016/j.future.2022.12.028
45. Lever, J. (2016) 'Classification evaluation: it is important to understand both what a classification metric expresses and what it hides', *Nature Methods*, 13, p. 603+. https://link.gale.com/apps/doc/A459507798/HRCA?u=anon~f33228a3&sid=googleScholar&xid=ceaf5104.
46. Li G, Jung JJ (2023) Deep learning for anomaly detection in multivariate time series: Approaches, applications, and challenges. Inf Fusion 91:93–102. https://doi.org/10.1016/j.inffus.2022.10.008
47. Li Z et al (2022) SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities. IEEE Transact Dependable Secure Comput 19(4):2244–2258. https://doi.org/10.1109/TDSC.2021.3051525
48. Liao Y, Vemuri VR (2002) Use of K-Nearest Neighbor classifier for intrusion detection. Comput Secur 21(5):439–448. https://doi.org/10.1016/S0167-4048(02)00514-X
49. Lorena AC et al (2019) How complex is your classification problem? a survey on measuring classification complexity. ACM Comput Surv (CSUR) 52(5):1–34
50. Macià N et al (2013) Learner excellence biased by data set selection: A case for data characterisation and artificial data sets. Pattern Recogn 46(3):1054–1066
51. Maciá-Fernández G et al (2018) UGR'16: A new dataset for the evaluation of cyclostationarity-based network IDSs. Comput Secur 73:411–424. https://doi.org/10.1016/j.cose.2017.11.004
52. Mao X et al (2019) Extractive summarization using supervised and unsupervised learning. Expert Syst Appl 133:173–181. https://doi.org/10.1016/j.eswa.2019.05.011
53. Marins MA et al (2018) Improved similarity-based modeling for the classification of rotating-machine failures. J Frankl Inst 355(4):1913–1930. https://doi.org/10.1016/j.jfranklin.2017.07.038
54. Meidan Yair, B.M.M.Y.M.Y.B.D.A. and Shabtai, A. (2018) 'detection_of_IoT_botnet_attacks_N_BaIoT'.
55. Miles, J. (2005) 'R-squared, adjusted R-squared', *Encyclopedia of statistics in behavioral science* [Preprint].
56. Min, H. *et al.* (2024) 'Toward interpretable anomaly detection for autonomous vehicles with denoising variational transformer', *Engineering Applications of Artificial Intelligence*, p. 107601. https://doi.org/10.1016/j.engappai.2023.107601.
57. Molan M et al (2023) RUAD: Unsupervised anomaly detection in HPC systems. Future Generation Comput Syst 141:542–554. https://doi.org/10.1016/j.future.2022.12.001
58. Morán-Fernández L, Bolón-Canedo V, Alonso-Betanzos A (2017) 'Can classification performance be predicted by complexity measures? A study using microarray data', Knowledge and Information Systems 51:1067–1090
59. Moustafa, N. and Slay, J. (2015) '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, pp. 1–6. https://doi.org/10.1109/MilCIS.2015.7348942.
60. Nguyen D-T, Le K-H (2023) The robust scheme for intrusion detection system in Internet of Things. Internet of Things 24:100999. https://doi.org/10.1016/j.iot.2023.100999
61. Okun O, Priisalu H (2009) Dataset complexity in gene expression based cancer classification using ensembles of k-nearest neighbors. Artif Intell Med 45(2–3):151–162
62. Pham, C. *et al.* (2014) '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, pp. 13–24. https://doi.org/10.1109/DSN.2014.19.
63. Popov, S., Morozov, S. and Babenko, A. (2020) 'Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data', in *International Conference on Learning Representations*. https://openreview.net/forum?id=r1eiu2VtwH.
64. Potvin PJ, Schutz RW (2000) Statistical power for the two-factor repeated measures ANOVA. Behav Res Methods Instrum Comput 32(2):347–356
65. Rachmawati SM et al (2023) Digital twin-enabled 3D printer fault detection for smart additive manufacturing. Eng Appl Artif Intell 124:106430. https://doi.org/10.1016/j.engappai.2023.106430
66. Rajadurai H, Gandhi UD (2022) A stacked ensemble learning model for intrusion detection in wireless network. Neural Comput Appl 34(18):15387–15395. https://doi.org/10.1007/s00521-020-04986-5
67. Randhawa RH et al (2024) Deep reinforcement learning based Evasion Generative Adversarial Network for botnet detection. Future Generation Comput Syst 150:294–302. https://doi.org/10.1016/j.future.2023.09.011
68. Ring M, Wunderlich S, Grüdl D, Landes D, Hotho A (2017) Flow-based benchmark data sets for intrusion detection. In: Proceedings of the 16th European conference on cyber warfare and security. ACPI, pp 361–369
69. Ring M et al (2019) A survey of network-based intrusion detection data sets. Comput Secur 86:147–167. https://doi.org/10.1016/j.cose.2019.06.005
70. Rivolli A et al (2022) Meta-features for meta-learning. Knowl-Based Syst 240:108101
71. Rodríguez P et al (2018) Beyond one-hot encoding: Lower dimensional target embedding. Image Vision Comput 75:21–31. https://doi.org/10.1016/j.imavis.2018.04.004
72. Saied M, Guirguis S, Madbouly M (2024) Review of artificial intelligence for enhancing intrusion detection in the internet of things. Eng Appl Artif Intell 127:107231. https://doi.org/10.1016/j.engappai.2023.107231
73. Sathya, R. and Abraham, A. (2013) 'Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification', *International Journal of Advanced Research in Artificial Intelligence*, 2(2). https://doi.org/10.14569/IJARAI.2013.020206.
74. Sharafaldin, I., Habibi Lashkari, A. and Ghorbani, A.A. (2018) '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, pp. 108–116. https://doi.org/10.5220/0006639801080116.
75. Sharmila BS, Nagapadma R (2023) Quantized autoencoder (QAE) intrusion detection system for anomaly detection in resource-constrained IoT devices using RT-IoT2022 dataset. Cybersecurity 6(1):41. https://doi.org/10.1186/s42400-023-00178-5
76. Shin, H.-K. *et al.* (2020) 'HAI 1.0: HIL-based Augmented ICS Security Dataset', in *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20)*. USENIX Association. https://www.usenix.org/conference/cset20/presentation/shin.
77. Shin Hyeok-Ki; Lee, W.C.S.Y.J.-H. and Min, B.-G. (2023) 'HAI security datasets'. https://github.com/icsdataset/hai.
78. Shiravi A et al (2012) Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Comput Secur 31(3):357–374. https://doi.org/10.1016/j.cose.2011.12.012
79. Shwartz-Ziv R, Armon A (2022) Tabular data: deep learning is not all you need. Inf Fusion 81:84–90. https://doi.org/10.1016/j.inffus.2021.11.011
80. Smith DJ, Simpson KGL (2020) The safety critical systems handbook: a straightforward guide to functional safety: IEC 61508 (2010 Edition), IEC 61511 (2015 edition) and related guidance. Butterworth-Heinemann

81. Souza, M.A. *et al.* (2024) 'A dynamic multiple classifier system using graph neural network for high dimensional overlapped data', *Information Fusion*, 103, p. 102145. https://doi.org/10.1016/j.inffus.2023.102145.
82. Tavallaee, M. *et al.* (2009) 'A detailed analysis of the KDD CUP 99 data set', in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*. IEEE, pp. 1–6. https://doi.org/10.1109/CISDA.2009.5356528.
83. Tsai C-F, Sung Y-T (2020) Ensemble feature selection in high dimension, low sample size datasets: parallel and serial combination approaches. Knowledge-Based Systems 203:106097. https://doi.org/10.1016/j.knosys.2020.106097
84. Urbanowicz RJ et al (2018) Benchmarking relief-based feature selection methods for bioinformatics data mining. J Biomed Inform 85:168–188
85. Wolpert DH (1996) The lack of a priori distinctions between learning algorithms. Neural Comput 8(7):1341–1390
86. Xia P, Zhang L, Li F (2015) Learning similarity with cosine similarity ensemble. Inf Sci 307:39–52
87. Xu Z, Saleh JH (2021) Machine learning for reliability engineering and safety applications: Review of current status and future opportunities. Reliab Eng Syst Saf 211:107530. https://doi.org/10.1016/j.ress.2021.107530
88. Yu, L. and Liu, H. (2003) 'Feature selection for high-dimensional data: A fast correlation-based filter solution', in *Proceedings of the 20th international conference on machine learning (ICML-03)*, pp. 856–863.
89. Zar JH (1972) Significance testing of the Spearman rank correlation coefficient. J Am Stat Assoc 67(339):578–580
90. Zhang S et al (2021) A temporal LASSO regression model for the emergency forecasting of the suspended sediment concentrations in coastal oceans: accuracy and interpretability. Eng Appl Artif Intell 100:104206. https://doi.org/10.1016/j.engappai.2021.104206
91. Zhao X et al (2015) A two-stage feature selection method with its application. Comput Electr Eng 47:114–125
92. Zhao, Y., Nasrullah, Z. and Li, Z. (2019) 'PyOD: A Python Toolbox for Scalable Outlier Detection', *Journal of Machine Learning Research*, 20(96), pp. 1–7. http://jmlr.org/papers/v20/19-011.html.
93. Zoppi, T., *et al.* (23AD) 'Anomaly Detectors for Self-Aware Edge and IoT Devices', in *2023 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE.
94. Zoppi T, Ceccarelli A, Puccetti T, Bondavalli A (2023) Which algorithm can detect unknown attacks? Comparison of supervised, unsupervised and meta-learning algorithms for intrusion detection. Comput Secur 127:103107
95. Zoppi T, Ceccarelli A, Bondavalli A (2019) MADneSs: a multilayer anomaly detection framework for complex dynamic systems. IEEE Transactions on Dependable and Secure computing. 18(2):796–809
96. Zoppi T, Ceccarelli A, Bondavalli A (2021) Unsupervised algorithms to detect zero-day attacks: strategy and application. IEEE Access 9:90603–15