Hindawi
Complexity
Volume 2019, Article ID 6278908, 16 pages
https://doi.org/10.1155/2019/6278908

WILEY | Hindawi

Research Article

Effect of the Sampling of a Dataset in the Hyperparameter Optimization Phase over the Efficiency of a Machine Learning Algorithm

Noemí DeCastro-García,[1] **Ángel Luis Muñoz Castañeda**,[2] **David Escudero García**,[2] **and Miguel V. Carriegos**[1]

[1]*Departamento de Matemáticas, Universidad de León, Campus de Vegazana s/n, 24071 León, Spain*
[2]*Research Institute on Applied Sciences in Cybersecurity, Universidad de León, Campus de Vegazana s/n, 24071 León, Spain*

Correspondence should be addressed to Noemí DeCastro-García; ncasg@unileon.es

Received 7 December 2018; Accepted 17 January 2019; Published 4 February 2019

Guest Editor: Fernando Sánchez Lasheras

Selecting the best configuration of hyperparameter values for a Machine Learning model yields directly in the performance of the model on the dataset. It is a laborious task that usually requires deep knowledge of the hyperparameter optimizations methods and the Machine Learning algorithms. Although there exist several automatic optimization techniques, these usually take significant resources, increasing the dynamic complexity in order to obtain a great accuracy. Since one of the most critical aspects in this computational consume is the available dataset, among others, in this paper we perform a study of the effect of using different partitions of a dataset in the hyperparameter optimization phase over the efficiency of a Machine Learning algorithm. Nonparametric inference has been used to measure the rate of different behaviors of the accuracy, time, and spatial complexity that are obtained among the partitions and the whole dataset. Also, a level of gain is assigned to each partition allowing us to study patterns and allocate whose samples are more profitable. Since Cybersecurity is a discipline in which the efficiency of Artificial Intelligence techniques is a key aspect in order to extract actionable knowledge, the statistical analyses have been carried out over five Cybersecurity datasets.

1. Introduction

A Machine Learning (ML) solution for a classification problem is effective if it works efficiently in terms of accuracy and the required computational cost. The improvement of the first factor is faced on by several points of view that could affect to the second one in different forms.

The simplest way to get a ML model with a good accuracy is by testing and comparing different ML algorithms for the same problem and choosing, finally, the one that performs better. However, it is clear that, for instance, a decision tree model does not require, in general, as much computational time and memory to be trained as a Multilayer Perceptron. So, we will need to adjust the achieved accuracy with the available resources.

Another usual effective approach to reach a high accuracy is working with large training datasets. Nevertheless, this solution is limited because of the associated computational cost (obtaining and storing the data, cleaning and transformation processes, and learning from the data). A possible alternative to the mentioned problem is to reduce the training without losing too much information [1, 2]. However, these kinds of solutions used to need an expensive data preprocessing phase.

The research related to this aspect, in addition to usual filtering the data, is focused on how to optimize the training set, and not only reduce it. The progressive sampling method shows that the performance with random samples with determined sizes is equal or more effective than working with the entire dataset [3]. Also, this solution used to be perfected with specifications about the classes' distribution, the selection of the samples, or the treatment of unbalanced datasets [4–6].

On the other hand, tuning hyperparameters of a ML algorithm is a critical aspect of the model training process that is considered the best practice for obtaining a successful Machine Learning application [7]. The Hyperparameters Optimization (HPO) problem requires a deep understanding of the ML model at hand due to the hyperparameters values settings and their effectivity, depending strongly on the ML algorithm, and the type of hyperparameter, discrete or continuous values. Also, it is a very costly process due to the large number of possible combinations to test and the needed resources to carry out the computations. Excluding the human expert of the process and minimizing the hyperparameter configurations to test are the underlying ideas in the automatic HPO.

Given a supervised ML algorithm, the continuous HPO is usually solved by gradient descent-based methods [8–10]. In the discrete case, not all optimization methods are suitable. The most common approaches to the HPO problem in the discrete case can be divided into two: Bayesian [11] and decision-theoretic methods. However, there are other optimization algorithms that are applied to the problem of hyperparameter values selection. This is the case, for instance, of the derivative-free optimization, the genetic algorithms such as Covariance Matrix Adaptation Evolutionary Strategies (CMA-ES), or the simplex Nelder-Mead (NM) method [12–14]. In addition, continuous optimization methods, such as the Particle Swarm (PS) used for the ML algorithms of least-squares support vector machines [15–17], could be applied over discrete HPO problems [18].

Bayesian HPO algorithms balance the exploration process for finding promising hyperparameter configurations and the exploitation of the setting configuration in order to obtain always better results or gain more information with each test [19]. Consequently, Bayesian HPO are serialized methods that are difficult to parallelize, but they usually can find better combinations of hyperparameters in a few iterations. One of the most important Bayesian HPO methods is the Model-Based Optimization (SMBO). In the SMBO techniques, the way to construct the surrogate function to model the error distribution provides different methods. There are those that use Gaussian Process (GP) [20] or tree-based algorithms such as the Sequential Model Automatic Configuration (SMAC) or the Tree Parzen Estimators (TPE) method [21, 22]. On the other hand, the decision-theoretic approaches are based on the idea to search combinations of hyperparameter in the hyperparameter space, computing their accuracy, and finally pick the one that performed the best. If we test over a fixed domain of hyperparameters values, we have the *grid search*. More effective than grid search, even than some Bayesian optimization techniques, is the random sampling of different choices Random Search (RS) [23]. It is easy to implement, independent of previous knowledge, and easily parallelizable. Other optimization approaches also have reached very good results when applied to the selection of hyperparameter values. This is the case of research of [24] where evolutionary computation is used over deep neural networks.

Although applying HPO algorithms on a ML model reflects a great improvement in the results' quality of the models' accuracy, we cannot overlook the computational complexity to implement these techniques. It is a critical issue because obtaining a good performance by applying HPO could require generations' samples, several function evaluations, and expensive computational resources. For example, the GP methods usually require a high number of iterations. Likewise, some derivative-free optimizations behave poorly in hyperparameter optimization problems because the optimization target is smooth [25]. In addition, some ML algorithms such as the neural networks are especially delicate because the number of possible values for the hyperparameters grows exponentially with the number of hidden layers. Other HPO algorithms are designed taking into account these limitations.

Recently, a Radial Basis Function (RBF) has been proposed as a deterministic surrogate model to approximate the error function of the hyperparameters through dynamic coordinate search that requires fewer evaluations in Multilayer Perceptron (MLP) and convolutional neural networks [26]. In [27], the Nelder-Mead and coordinate-search (derivative-free) methods are tested over deep neural networks, presenting more efficient numerical results than other well-known algorithms. Another option is to try to accelerate the algorithm. In [28], one way to implement the scheme Successive Halving (SH) is developed. The main ingredient behind SH is based on the observation that most of the HPO algorithms are iterative which suggests that stopping the algorithm when testing with a set of hyperparameters is not giving good results can be a good option.

Examples of this application are the accelerated RS version (2x) [29, 30] or Hyperband [31], which is based on the bandit-based approach. In this problem, a fixed finite set of resources must be distributed between different choices in a way that maximizes their expected gain. One recent example is developed in [19], where an algorithm for optimization of discrete hyperparameters based on compressed sensing is introduced, *Harmonica*.

We can also find studies about the effect of these HPO methods in the efficiency of the ML algorithms comparing different methods [32], or the possible options that can be tuned when a HPO algorithm runs over a dataset such as the number of iterations, number of hyperparameters, type of optimization, etc. [24, 33]. As far as we know, in the above context, analyzing the efficiency of the automatic HPO methods for specific supervised ML problems in terms of the size of the used dataset is needed.

The goal in this article is to carry out an empirical statistical analysis about the effect of the factor size of datasets used in the stage of the HPO in the performance of several ML algorithms. The studied response variables are the main issues in the efficiency of the algorithm: quality and dynamic complexity [34]. Regarding the quality, we take into account the most used metric for the goodness of a ML model, that is, the Accuracy. The others variables are the time complexity and the spatial complexity (amount of memory it requires for execution with a given input) due to these issues usually are influent limited resources. The underlying idea is to allocate the proportion of the whole dataset that maintains or improves the quality of the accuracy of the obtained

ML models and optimizing the dynamic complexity of the algorithms.

The research questions that will be studied are the following:

RQ1: Given a dataset, are there statistically meaningful differences in the performance (accuracy, time, and special complexity) of a ML algorithm when the HPO method it is applied to samples of different size of the dataset?

In the case that we obtain a positive answer, we follow to the next questions.

RQ2: Which are the effect and the behavior of each HPO algorithm depending on the dataset's size? In which cases we can gain efficiency if the HPO algorithm operates over small samples?

RQ3: Are the above results reliable? That is, are the results consistent for different HPO algorithms and different datasets?

In order to answer the research questions formulated above, an experiment has been carried out with different publicly available datasets about learning some tasks regarding cybersecurity events. Cybersecurity is a challenging research area due to the sophistication and the amount of kind of Cybersecurity attacks, which, in fact, increase very fast as time goes by. In this framework, the traditional tools and infrastructures are not useful because we deal with big data created with a high velocity, and the solutions and predictions must be faster than the threats. Artificial Intelligence and ML analytics have turned out in one of the most powerful tools against the cyberattackers (see [35–41]), but obtaining actionable knowledge from a database of Cybersecurity events by applying ML algorithms usually is a computationally expensive task for several reasons. A database of Cybersecurity contains, in general, a huge amount of dynamical, and unstructured but highly correlated and connected data, so we need to deal with some costly aspects of the quality of the data such as noise, trustworthiness, security, privacy, heterogeneity, scaling, or timeliness [42–44]. Also, the information is highly volatile, so the key issue is to get the profit as faster as possible, with the best possible performance and the smallest cost of resources and time. Then, the study of the complexity of applying Data Science over databases of Cybersecurity is an emergent and necessary field in order to increase confidence and social profit from automatizing processes and develop prescriptive policies to prevent and react to incidents faster and more secure.

Experimental analyses have been carried out in order to investigate the possible statistical differences over the efficiency of the Machine Learning algorithms Random Forest (RF), Gradient Boosting (GB), and MLP among using different sizes of samples in several HPO selection algorithms. We have used nonparametric statistical inference because, in an experimental design in the field of computational intelligence, these types of techniques are very useful to analyze the behavior of a method with respect to a set of algorithms [45]. In addition and based on the results of the above tests, we have assigned a profit level for each size of the whole dataset in terms of the response variables: accuracy, time, and spatial complexity. Finally, we have discussed the observed patterns and obtained results.

The paper is structured as follows. In Section 2, we describe the problem definition. In Section 3 we develop the Materials and Methods. This section includes the analyzed selected methods, both HPO and ML algorithms with the libraries or tools that we have used. Also, the tested datasets and the sampling of partitions have been explained, as well as the statistical analyses that have been carried out. In Section 4, the results and the discussion are included. Finally, our conclusions and references are given.

2. Problem Definition

Let $\mathscr{P}$ be a distribution function. A Machine Learning algorithm, A, is a functional that maps each data set containing i.i.d. samples from $\mathscr{P}$, D^{train}, to a function $f_{A,D^{train}} \coloneqq A(D^{train})$ that belongs to certain space of functions and that minimizes a fixed expected loss $\mathscr{L}(D^{train}, f_{A,D^{train}})$. Usually one has two more ingredients in this general framework. On one hand, the target space of functions of the algorithm depends on certain parameters, $\lambda = (\lambda_1, \ldots, \lambda_n)$, that might take discrete or continuous values and that has to be fixed before applying the algorithm. In order to make explicit the dependency on λ, we will use the notation A_λ to refer to the algorithm and f_λ to refer to the target functions of A_λ. On the other hand, another data set obtained from $\mathscr{P}$, D^{test}, is given and serves to evaluate the loss, $\mathscr{L}(D^{test}, f_{A_\lambda,D^{train}})$, of the function provided by the algorithm over data independent from D^{train}.

Let Λ be the hyperparameter space, that is, the space in which λ takes values, and fix both the train and the test data sets. Denote by $D \coloneqq D^{train} \cup D^{test}$ the union of both data sets. In this situation the only data that remains free in $\mathscr{L}(D^{test}, f_{A_\lambda,D^{train}})$ are the hyperparameters $\lambda = (\lambda_1, \ldots, \lambda_n)$. Assume further that we are dealing with a classification Machine Learning problem, so that we can take $\mathscr{L}$ to be the error rate, that is, one minus the cross-validation value. In this situation one can define the following function:

$$\begin{aligned} \Phi_{A,D} : \Lambda &\longrightarrow [0,1] \\ \lambda &\longmapsto \operatorname{mean}_{D^{test}} \mathscr{L}\left(D^{test}, f_{A_\lambda,D^{train}}\right) \end{aligned} \tag{1}$$

The HPO problem consists on minimizing $\Phi_{A,D}$ and, therefore, a HPO algorithm is a procedure that tries to reach $\lambda^* \coloneqq \min_\lambda(\operatorname{mean}_{D^{test}} \mathscr{L}(D^{test}, f_{A_\lambda,D^{train}}))$.

Usually, in practice, one has a dataset D that is split into two parts, D_1, D_2, one for the optimization of the hyperparameters of the Machine Learning model and another one for training (and testing) the Machine Learning model with the given hyperparameters. The goal of the article is focused on how the size of the dataset in the HPO phase influences the performance of classifier given at the output of the training phase.

Suppose we have a ML model A, a HPO algorithm H, and a dataset D. The dataset D is split into different partitions randomly. We denote by $P_j(D)$ the partition j of the dataset D, and we assume that the size of $P_j(D)$ is smaller than the size of $P_l(D)$ for $j < l$. Then, applying H to the problem defined by $\Phi_{A,P_j(D)}$, we find optimal hyperparameters λ^j that can be used

Table 1: HPO algorithms. The star symbol * means that the chosen algorithms have needed some minor modifications.

Name	Reference	Ready?	Python minimal version	Smart	Library
PS	[15, 16]	√	2.7 y 3	X	[18]
TPE	[21]	√*	2.7 y 3	X	[51]
CMA-ES	[13]	√	2.7 y 3	X	[52]
NM	[14]	√	2.7 y 3	√	[52]
RS	[23]	√	2.7 y 3	X	[52]
SMAC	[22]	√	3	X	[53]

over D^{train} to create a classifier that will be tested on D^{test}. We denote the cross-validation value obtained in this last process by Acc^j. Also, the train time and the total time of the process and the spatial complexity spent will be collected (denoted by TC^j and SC^j respectively) for the analysis.

The study that will be done is statistical, so we will consider several datasets as well as many Machine Learning algorithms. Different state-of-the-art HPO algorithms will be considered and applied to every possible combination of Machine Learning models and datasets in order to compare them.

3. Materials and Methods

Experiments were conducted testing eight HPO methods over five cybersecurity datasets for three ML algorithms.

3.1. HPO Methods and ML Algorithms. As we mentioned above, we have evaluated the efficiency of the three well-known classifiers and predictor Machine Learning techniques algorithms: RF, GB, and MLP, commonly used in classification and prediction problems. The library used is [46].

Ensembles methods are commonly used because are based on the underlying idea that many different joint predictors will perform in a better way than any single predictor alone. Ensembling techniques could be divided into bagging and boosting methods.

First ones build many independent learners that are combined with average methods in order to give a final prediction. These handle overfitting and reduce the variance. The most known example of bagging ensemble methods is the RF. An RF is a classifier consisting of a chain of decision tree algorithms. Each tree is constructed by applying an algorithm to the training set and an additional random vector that is sampled via bootstrap resampling, so the trees will run and give independent results (see [47]). The scikit-learn implementation, RandomForestClassifier, combines classifiers by calculating an average of their probabilistic prediction, in contrast to the original publication. In the case of RF, we have two main hyperparameters: the number of decision trees that we should use and the maximum depth for each of them.

Second ones, Boosting, are ensemble techniques in which the predictors are made sequentially, learning from the mistakes of the previous predictor in order to optimize the subsequent learner. It usually takes less time/iterations to reach close to actual predictions, but we have to choose the stopping criteria carefully. These reduce bias and variance and can with the overfitting. One example of the most common boosting methods is GB. The library that is used is the GradientBoostingClassifier, and we tune the discrete hyperparameters that are the number of predictors and the maximum depth of them.

On the other hand, an artificial neural network is a model that is organized in layers (input layer, output layer, and hidden layers). An MLP is a modification of the standard linear perceptron where multiple layers of connected nodes are allowed. The standard algorithm for training a MLP is the backpropagation algorithm (see [48, 49]). Class MLPClassifier in Scikit-learn implements MLP training algorithms for this ML model. By default, MLPClassifier has only one hidden layer with 100 neurons, the activation function for the hidden layer is f (x) = max(0, x), and the solver for weight optimization is "Adam" [50]. In this study, we will allow two hidden layers and we will run the number of neurons in each of these hidden layers.

In Table 1, the selected HPO algorithms to study are developed. Also, the references related to each one are given, as well as the minimal version of Python for which these algorithms work. In addition, we highlight if they are smart: that it is, if they stop by themselves in contrast to a number of iterations (trials) must be proposed.

3.2. Datasets. The choice of datasets selected for the experiments was motivated by different reasons: available in public servers, diversity in the number of instances, classes, and features, and relating to cybersecurity. The set of datasets $\mathscr{D} = \{D_1, D_2, D_3, D_4, D_5\}$ is described in Table 2.

Regarding the transformation of features data of treatable datasets, this has been performed manually by Python.

Dataset D_1 is a collection of spam (advertisements for products/web sites, make money fast schemes, chain letters, pornography, etc.) and nonspam e-mails whose purpose is to construct spam filters. Dataset D_2 is a database obtained by a real-time localization system for an autonomous robot with examples that are constructed with simulated attacks (Denial of Service and Spoofing) and nonattacks. Dataset D_3 is about de detection of Phishing websites. D_4 is a data set suggested to solve some of the inherent problems of the well-known KDD′99 data set. This has been treated by the approach given in [59, 60]. Finally, D_5 contains transactions made by credit cards. This unbalanced collection of data includes examples labeled as fraudulent or genuine. Also, the features are the result of a PCA transformation.

Table 2: Description of the set of datasets $\mathscr{D}$.

Dataset	Name	Instances	Features	Classes	Reference
D_1	Spambase	4601	57	2	[54]
D_2	Robots in RTLS	6422	12	3	[55]
D_3	Phishing websites	11055	30	2	[56–58]
D_4	Intrusion Detection (NSL-KDD)	148517	39	5	[59, 60]
D_5	Credit Card Fraud Detection	284807	30	2	[61]

Table 3: Description of the partitions of the set of datasets $\mathscr{D}$.

Dataset	P_1	P_2	P_3	P_4
D_1	383	766	2300	4601
D_2	535	1070	3211	6422
D_3	921	1842	5527	11055
D_4	12376	24752	74258	148517
D_5	23733	47467	142403	284807

Also, the datasets have different number of instances as well as features and number of classes of the target variable.

3.3. Sampling and Collection of Data. For each dataset D_i, four partitions, $\{P_j(D_i)\}_{j=1,\ldots,4}$, have been created in a random way. These correspond to the proportions that are obtained when multiplied by 1/2, 1/6, 1/12 the whole set ($\mathscr{P} = \{P_1 = 8,3\%, P_2 = 16,3\%, P_3 = 50\%, P4 = D_i\}$). We denote by $P_j(D_i)$ the partition on the dataset D_i where $i = 1,\ldots,5$ and $j = 1,2,3,4$. These proportions have been chosen due to the fact that, in this way, we can deal with an amount of instances of different order of magnitude, ($10^2, 10^3, 10^4$, and 10^5, having similar orders in the same partition of each dataset; see Table 3.

On the other hand, we fix once and for all a partition $D_i^{train} \cup D_i^{valid} = D_i$ for each D_i into train data (80%) and validation data (20%), as well as a partition with the same proportion for each partition $P_j(D_i)$.

Finally, in order to build response variables to measure the goal of the study, we apply each $H_k \in \mathscr{H}$ over the all partitions, $P_j(D_i)$, obtaining the hyperparameter configuration $\lambda_{i,j}^k$. Then, the learning algorithm with the obtained hyperparameter configuration is run over D_i^{train} to construct a classifier that is validated over D_i^{valid}. At this step, we collect the accuracy obtained. This scenario is repeated 50 iterations. Also, the total time and spatial complexity of all process are stored. Then we create three response variables. We denote by $Acc_{i,j}^k$ the 50×1 array where the m-th component is the accuracy of the predictive model tested on D_i^{valid} that was trained over D_i^{train} with the hyperparameters $(\lambda_{i,j}^k)_m$ at the m-th iteration. We can measure the time and the spatial complexity used along this process and collect these data in two 50×1 arrays, $TC_{i,j}^k$, and $SC_{i,j}^k$, respectively.

The time complexity, measured in seconds, is the sum of the time needed by the HPO algorithm for finding the optimum hyperparameters and the time needed by the ML algorithm for the training phase. The spatial complexity, measured in Kb, is defined as the maximum of use of memory along the HPO algorithm run, including the internal structures of the algorithm as well as train and test datasets load.

3.4. Technical Specifications. The analyses have been carried by the authors at high-performance computing facilitated by SCAYLE (www.scayle.es) over HP ProLiant SL270s Gen8 SE, with 2 processors Intel Xeon CPU E5-2670 v2 @ 2.50GHz with 10 cores each one, and 128 GB of RAM memory. They are equipped with 1 hard disk of 1TB and cards Infiniband FDR 56Gbps.

The analyses script has been implemented in Python language. Python uses an automated manager system of memory called garbage collector that releases the unused memory space. This phenomenon might be nondeterministic and certain fluctuations shown in the results may be due to not releasing the memory in that case.

It is worth noting that different technical tools (either software or hardware) could affect to the data about time and spatial complexity that have been collected. This is a fact that should be taken into account if we want to measure the effect of data sizes over the response variables in absolute terms, that is, over a single HPO algorithm. However, the influence of the technical specifications in the response variables is not a relevant factor in this study. This is a comparative study in which all the measures are collected under the same conditions, and the possible effect of the technical elements on the data is the same in each experiment. It is expected that the same behavior of the comparative encountered patterns will appear with other technical characteristics.

3.5. Analysis. The aim of the study is to decide if the size of data is a factor that influences in the efficiency of a ML algorithm using a HPO method among partitions of the data.

We perform the following statistical analysis:

(1) First, for each level of the size, that is, the partitions P_j where $j = 1,2,3,4$, we study the normality of $Acc_{i,j}^k$

Table 4: Level of gains where ' =,<,>' denotes statistically meaningful equality and differences, *Me* represents the median, and $j < l$.

Level	Condition
9	If $Me(Acc_{i,j}^k) > Me(Acc_{i,l}^k)$ & $Me(SC_{i,j}^k) < Me(SC_{i,l}^k)$
8	If $Me(Acc_{i,j}^k) > Me(Acc_{i,l}^k)$ & $Me(SC_{i,j}^k) = Me(SC_{i,l}^k)$
7	If $Me(Acc_{i,j}^k) = Me(Acc_{i,l}^k)$ & $Me(SC_{i,j}^k) < Me(SC_{i,l}^k)$
6	If $Me(Acc_{i,j}^k) = Me(Acc_{i,l}^k)$ & $Me(SC_{i,j}^k) = Me(SC_{i,l}^k)$
5	If $Me(Acc_{i,j}^k) > Me(Acc_{i,l}^k)$ & $Me(SC_{i,j}^k) > Me(SC_{i,l}^k)$ & $\Delta_{j,t}(SC_i^k) < \Delta_{j,t}(Acc_i^k)$ & $\Delta_{j,t}(SC_i^k) < \Delta_{j,t}(TC_i^k)$
4	If $Me(Acc_{i,j}^k) = Me(Acc_{i,l}^k)$ & $Me(SC_{i,j}^k) > Me(SC_{i,l}^k)$ & $\Delta_{j,t}(SC_i^k) < \Delta_{j,t}(TC_i^k)$
3	If $Me(Acc_{i,j}^k) < Me(Acc_{i,l}^k)$ & $Me(SC_{i,j}^k) < Me(SC_{i,l}^k)$ & $\Delta_{j,t}(SC_i^k) > \Delta_{j,t}(Acc_i^k)$ & $\Delta_{j,t}(TC_i^k) > \Delta_{j,t}(Acc_i^k)$
2	If $Me(Acc_{i,j}^k) < Me(Acc_{i,l}^k)$ & $Me(SC_{i,j}^k) = Me(SC_{i,l}^k)$ & $\Delta_{j,t}(TC_i^k) > \Delta_{j,t}(Acc_i^k)$
1	If $Me(Acc_{i,j}^k) < Me(Acc_{i,l}^k)$ & $Me(SC_{i,j}^k) > Me(SC_{i,l}^k)$ & $\Delta_{j,t}(TC_i^k) > \Delta_{j,t}(Acc_i^k)$ & $\Delta_{j,t}(TC_i^k) > \Delta_{j,t}(SC_i^k)$
0	In another case

(resp., $TC_{i,j}^k$ and $SC_{i,j}^k$). Once it has been determined that these data do not follow a normal statistical distribution, we have performed nonparametric tests. Wilcoxon' test for two paired samples was conducted in order to decide whether there are statistically meaningful differences or not among $Acc_{i,1}^k$, $Acc_{i,2}^k$, $Acc_{i,3}^k$, and $Acc_{i,4}^k$ (resp., for $TC_{i,j}^k$ and $SC_{i,j}^k$) for all $D_i \in \mathscr{D}$, $H_k \in \mathscr{H}$ and for the three selected ML algorithms (RF, GB, and MLP). The choice of this test is due to the fact that the response variables that we compare are obtained by the application of the ML algorithms over the dataset D_i with the splits D_i^{train} and D_i^{valid}, but with the different setting $(\lambda_{i,\cdot}^k)^*$ and $(\lambda_{i,4}^k)^*$. The study has been carried out with a significance level $\alpha = 0.05$.

(2) At this point, we have applied the inference described over the accuracy, the time, and the spatial complexity. Then, we have obtained the p-values of each partition's comparison for each response variable, for each HPO method, and for each D_i. That is six comparisons for P_1VsP_4 along five datasets providing a total of 30 decisions about statistical equality or not for each ML algorithm (the same process for P_2VsP_4, P_3VsP_4, respectively). Then, we have computed the rate of p-values that provides statistical differences by comparison of partitions, by response variables, and in a total way.

(3) From the results obtained in Wilcoxon's tests we assign to each $P_j(D_i)$ a level of gain with regard to $P_4(D_i)$ for $j = 1, 2, 3$. The mapping is carried out according to Table 4 where $\triangle_{j,t}(Acc_i^k)$ denotes the rate of absolute difference of the median accuracy between operating with the model obtained through HPO on $P_j(D_i)$ and $P_4(D_i)$ (resp., spatial and time complexity). Namely,

$$\underset{j,t}{\triangle}\left(Acc_i^k\right) = \frac{\left|Me\left(Acc_{i,j}^k\right) - Me\left(Acc_{i,t}^k\right)\right|}{\min\left[Me\left(Acc_{i,j}^k\right), Me\left(Acc_{i,t}^k\right)\right]} \quad (2)$$

$$\underset{j,t}{\triangle}\left(SC_i^k\right) = \frac{\left|Me\left(SC_{i,j}^k\right) - Me\left(SC_{i,t}^k\right)\right|}{\min\left[Me\left(SC_{i,j}^k\right), Me\left(SC_{i,t}^k\right)\right]} \quad (3)$$

$$\underset{j,t}{\triangle}\left(TC_i^k\right) = \frac{\left|Me\left(TC_{i,j}^k\right) - Me\left(TC_{i,t}^k\right)\right|}{\min\left[Me\left(TC_{i,j}^k\right), Me\left(TC_{i,t}^k\right)\right]} \quad (4)$$

This correspondence is shown for each algorithm H_k and dataset D_i.

The design of Table 4 was done based on how many response variables show a positive efficiency rate when optimizing the hyperparameter values with a smaller partitions instead of the whole dataset. First, we have created nine levels of gain (9 is the highest level), each of them determined by the number of response variables in which we reach more efficient results prioritizing the accuracy against T.C. and S.C. Also, we have sorted the levels of gain from the best combination to the worst, dropping those combinations in which the loss is larger than the profit. Due to the Python garbage collector, the response variable with a more clear increasing trend should be the time complexity instead of spatial complexity. Then, we suppose that the inequality T.C.$(P_j(D_i))$ < T.C.$(P_4(D_i))$ holds always true.

(4) Finally, we compute the average of gain of the smaller partitions in a general way, per datasets, and ML algorithms. These results let us measure the reliability of the conclusions.

4. Results and Discussion

The first research question deals with whether the size of a partition used in the HPO phase influences in certain sense on the efficiency of the algorithm. Once the comparison's tests described in the above section have been done, we account, for each ML model, how many combinations show statistically significant differences across all the HPO methods and all the datasets. Although we find an influence on the response variables, we do not know whether this influence is positive or not. So, the second research question is focused on

Table 5: Statistically significant differences in each dimension for all HPO across all D_i of RF for each comparison between $P_j(D_i)$ and $P_4(D_i)$ and the total (%).

Combination	Accuracy	Time Complexity	Spatial Complexity	(Acc+T.C.+S.C)
P_1 Vs P_4	13/30 = 43.33%	30/30 = 100%	30/30 = 100%	73/90 = 81.11%
P_2 Vs P_4	12/30 = 40%	29/30 = 96.66%	30/30 = 100%	71/90 = 78.88%
P_3 Vs P_4	3/30 = 10%	29/30 = 96.66%	30/30 = 100%	62/90 = 68.88%
$(P_1 + P_2 + P_3)$ Vs P_4	28/90 = 31.11%	88/90 = 97.77%	90/90 = 100%	

Table 6: Statistically significant differences in each dimension for all HPO across all D_i of GB for each comparison between $P_j(D_i)$ and $P_4(D_i)$ and the total (%).

Combination	Accuracy	Time Complexity	Spatial Complexity	(Acc+T.C.+S.C)
P_1 Vs P_4	19/30 = 63.33%	29/30 = 96.66%	30/30 = 100%	78/90 = 86.66%
P_2 Vs P_4	20/30 = 66.66%	29/30 = 96.66%	29/30 = 96.66%	78/90 = 86.66%
P_3 Vs P_4	14/30 = 46.66%	28/30 = 93.33%	30/30 = 100%	72/90 = 80%
$(P_1 + P_2 + P_3)$ Vs P_4	53/90 = 58.88%	86/90 = 95.55%	89/90 = 98.88%	

Table 7: Statistically significant differences in each dimension for all HPO across all D_i of MLP for each comparison between $P_j(D_i)$ and $P_4(D_i)$ and the total (%)).

Combination	Accuracy	Time Complexity	Spatial Complexity	(Acc+T.C.+S.C)
P_1 Vs P_4	4/30 = 13.33%	29/30 = 96.66%	30/30 = 100%	63/90 = 70%
P_2 Vs P_4	5/30 = 16.66%	28/30 = 93.33%	30/30 = 100%	63/90 = 70%
P_3 Vs P_4	1/30 = 3.33%	26/30 = 86.66%	30/30 = 100%	57/90 = 63.33%
$(P_1 + P_2 + P_3)$ Vs P_4	10/90 = 11.11%	83/90 = 92.22%	90/90 = 100%	

the study of this differences and equalities. Next, we analyze the reliability of the results. Finally, we include an overview of the global results that are obtained.

4.1. Research Question 1. In the case of RF, the results included in Table 5 show that the obtained accuracy in the 31.11% of possible combinations between the smaller partitions and the whole dataset can be considered statistically different. However, the time and spatial complexity have a very high amount of statistically significant differences (97.77% and 100%, respectively). Hence spatial and time complexity are affected by the size of the dataset (as expected). But, on the other hand, the size of the dataset does not impact on the accuracy in a critical way.

Also, we can see that the partition $P_1(D_i)$ reaches the highest number of differences in the accuracies, as well as in time and spatial complexity, while the behavior of the partition $P_3(D_i)$ is the more statistically similar to the whole dataset. Note that, in the 90 % of the cases, the accuracy obtained with this partition can be considered equal to accuracy obtained with $P_4 = D_i$. Note that the behavior of the accuracies shows an increasing trend of similarity related to the increasing size of the partition.

In the case of GB, the results included in Table 6 show that the obtained accuracy in the 41.12% of possible combinations among the smaller partitions and the whole dataset can be considered statistically equivalent. However, the time and spatial complexity have a very high amount of statistically significant differences (95.55% and 98.88%, respectively). So, we can confirm that the dataset's size has an effect on these last dimensions in a strong way, and over the accuracy in a medium level.

Regarding the concrete partitions, we have a greater homogeneity of the results in the GB than in RF, although $P_3(D_i)$ remains as the partition with the most similar accuracy with respect to the whole dataset (53.34 %). If we take into account the three response variables the rate of differences is quite high, $P_3(D_i)$ being the most similar with an 80% of differences. Note that, in this case, there is no increasing trend of similarity in the accuracies as the size grows up.

Finally, in the case of MLP, the results included in Table 7 show that the obtained accuracies in the 88.89 % of possible combinations among the smaller partitions and the whole dataset can be considered statistically equal. Then, the accuracy is not being quite affected by the size of the dataset used in the HPO phase. However, the time and spatial complexities have a very high amount of statistically significant differences (92.22% and 100%, respectively).

It is worth noting that $P_3(D_i)$ has obtained 96.67% of equivalent accuracies, but the behavior of the similarity in the accuracies, sorted by the size of the partition, is not found either in this case.

In general, we have found a high effect of the dataset's size used in the HPO over the time and spatial complexity, for the three ML algorithms. In the case of the accuracy, the ensemble methods (RF and GB) show a medium effect

Table 8: Patterns of profit.

ML method	Pattern 1	Pattern 2	Pattern 3	Pattern 4
RF	NM	SMAC, RS	PS, TPE, CMA-ES	
GB	RS	NM, TPE, CMA-ES	SMAC, PS	
MLP		RS, SMAC	CMA-ES	PS, TPE, NM

Table 9: Average rate of statistically significant differences in each dimension for all HPO across all D_i for all ML algorithms and for each comparison between $P_j(D_i)$ and $P_4(D_i)$ (%).

Combination	Accuracy	Time Complexity	Spatial Complexity
P_1 Vs P_4	39.99%	97.77%	100%
P_2 Vs P_4	41.10%	95.55%	98.88%
P_3 Vs P_4	19.99%	92.21%	100%
$(P_1 + P_2 + P_3)$ Vs P_4	33.7%	95.18%	99.62%

(around a rate of 40%), while in the MLP, the level of effect is very low.

4.2. Research Question 2. In order to study in depth whether the considered effect is positive or negative when we work with smaller partitions, the evolution of the response variables as the size of the partition grows up is developed. We are going to study how are the encountered differences in each dimension of the efficiency.

In Figures 1, 2, and 3, the charts of the evolution of the behavior of the studied dimensions are shown for each dataset D_i and for RF, GB, and MLP, respectively.

Both the time and spatial complexity appear with an increasing trend, the first one being more highlighted. Also, the case of spatial complexity is more variable in the case of MLP than in the ensemble methods. So, in order to gain efficiency when tuning the hyperparameters with a smaller proportion of data, different levels are assigned according to Table 4. As we have mentioned above, the levels of profit are proposed in terms of the gain in the accuracy and spatial complexity due to the fact that the time complexity shows a clear increasing trend.

Note that, in general, is not true that the accuracy increases as the size of the partition used for the HPO phase does. This can be seen more clearly when the ML model is GB or MLP and, certainly, depends on the dataset. See, for instance, the three charts for D_2 and D_3 in Figures 1, 2, and 3.

The gain's averages obtained in RF, GB, and MLP are included in Figures 4, 5, and 6.

In all the studied ML algorithms, we can obtain a gain when smaller partitions are used to HPO. Also, we can clearly find four different patterns (see Table 8). The first and second one show that the partition P_2 obtains the highest and the lowest profit. The third one is an increasing trend from P_1 to P_3. In the case of MLP, we also detect another pattern that stabilizes the gain from P_2.

In the case of RF, the lowest level of gain is 4.5 and the maximum value is 7. The other ensemble method, GB, has obtained levels of profit between 2.5 and 8. Finally, the MLP algorithm shows values between 5 and 7.5. Then, the neural network is the algorithm in which the HPO phase performs better with smaller partitions, followed by RF and GB.

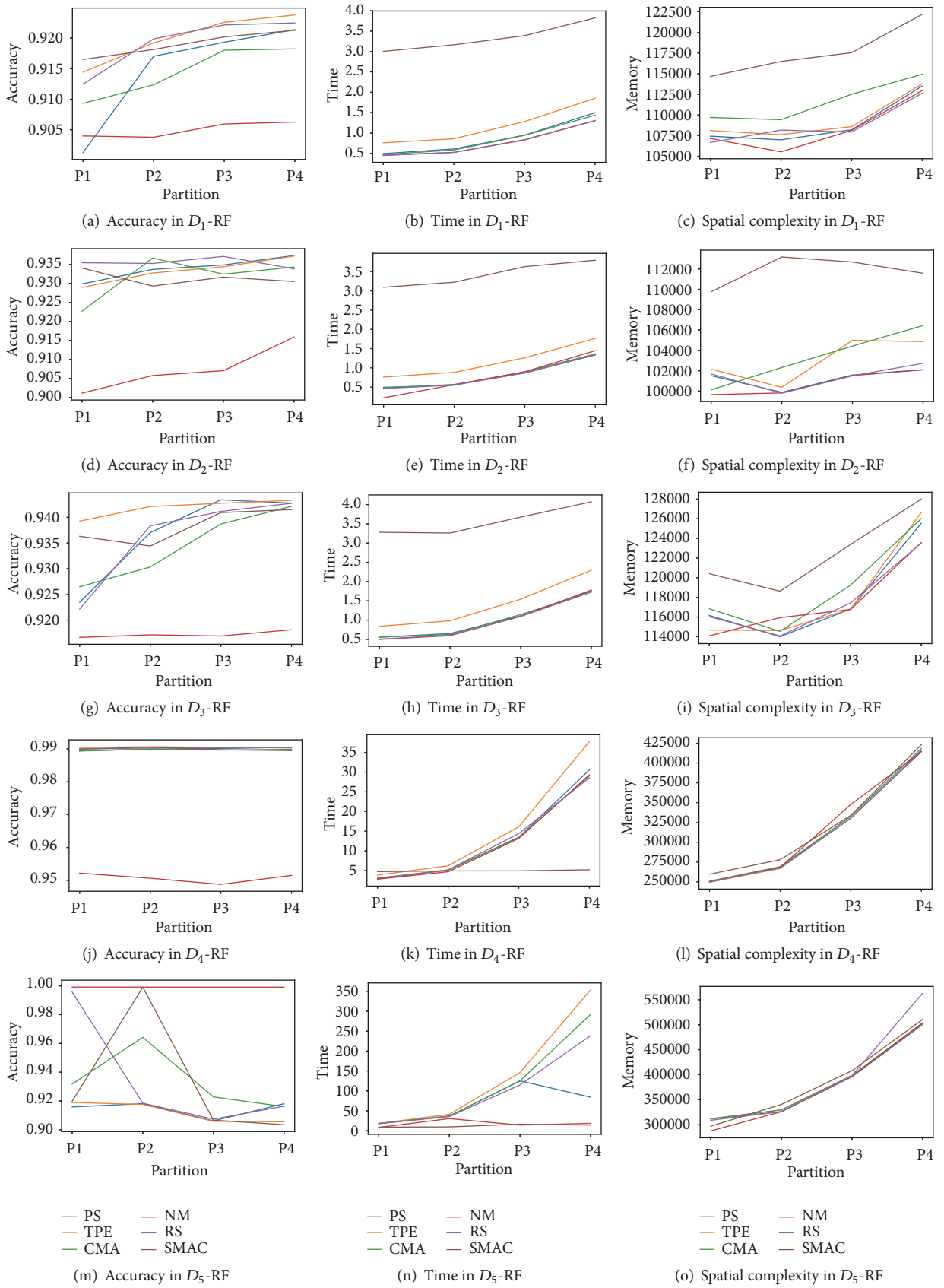
In addition, for all ML algorithms there is at least one HPO algorithm that obtains a profit of level greater than 6 in the smaller partitions P_1 and P_2, namely, the NM algorithm. In case of P_3, it is should be noted that the minimum level of gain is 5.5 for all HPO and ML techniques.

4.3. Research Question 3. If we compute the average of gain in each dataset, we obtain the results shown in Figure 7.

The averages of the profit level for RF are between 3.2 and 8.5, being the largest dataset D_5 the one in which we reach more efficiency working with smallest partition and showing a decreasing trend. On the other hand, the dataset D_3 is the one in which we get less gain of efficiency. The same behavior is presented in MLP up to a little loss of level, between 2.5 and 7.5. In the case of GB, the averages of the profit level are between 3.2 and 7.2, being the largest dataset D_4 the one in which we reach more efficiency working with smallest partition.

Finally, we can conclude that, in a general way, in all datasets we obtain efficiency optimizing the hyperparameter values with smaller partition, although the data were different in terms of features, number of instances, or classes of the target variable. So, the results are consistent and reliable.

4.4. Global Results. In Table 9 we include the global average rate of statistically significant differences in each dimension (see Tables 5, 6, and 7). As we have mentioned above, time and spatial complexity show an increasing trend directly related to the size of the partition. Therefore, the high rate of differences appearing in these variables in Table 9 is intuitive and expected. However, the behavior of the accuracy is different. We can observe that the accuracy that we get with smaller partitions is statistically equivalent to the obtained accuracy with the whole dataset in more than the 60% of the cases. This rate is greater than 80% for P3 (the 50% of the all dataset). This fact should be taken into account, especially in case of big data contexts, in order to optimize the available



(a) Accuracy in D_1-RF (b) Time in D_1-RF (c) Spatial complexity in D_1-RF

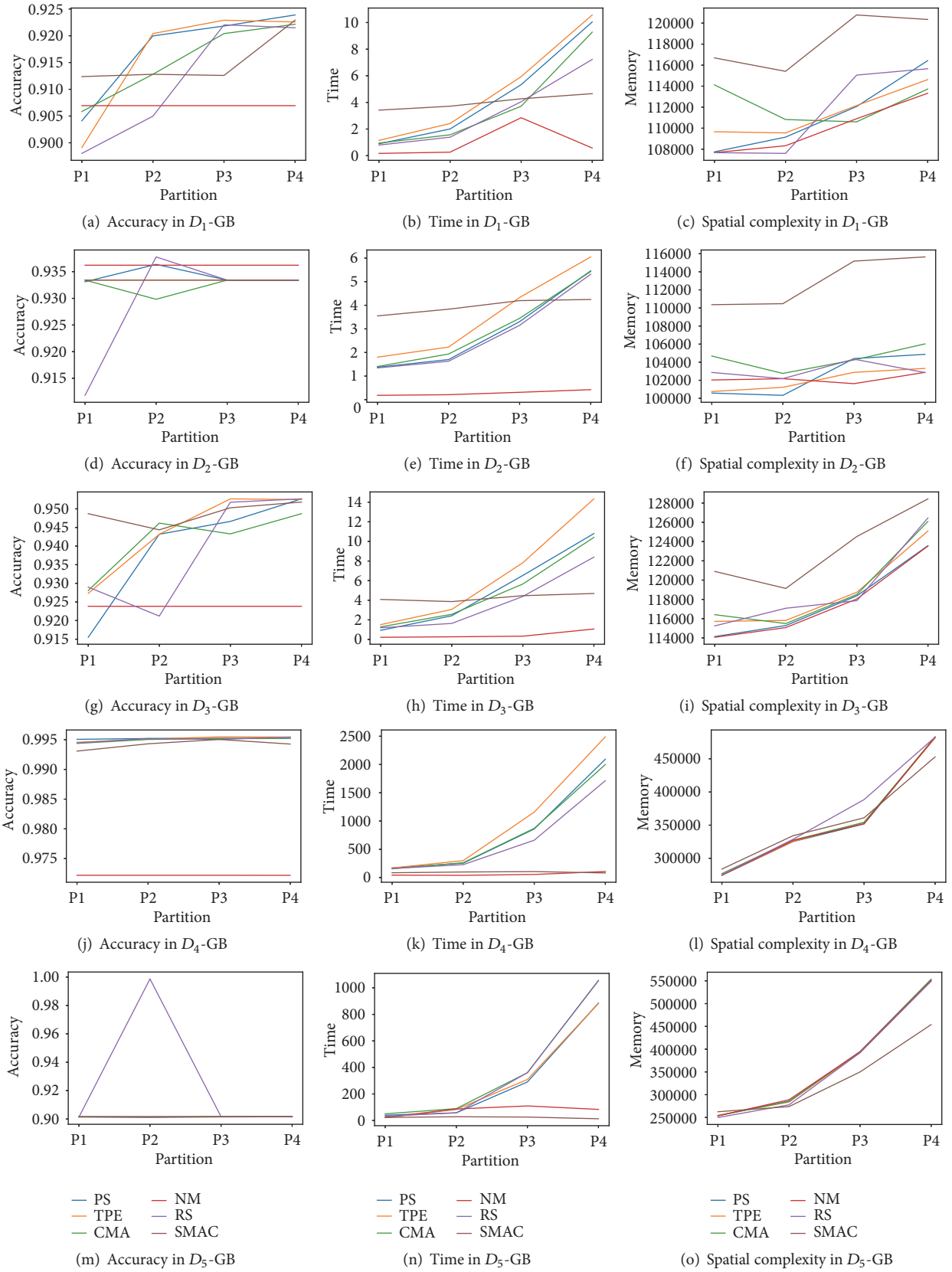
(d) Accuracy in D_2-RF (e) Time in D_2-RF (f) Spatial complexity in D_2-RF

(g) Accuracy in D_3-RF (h) Time in D_3-RF (i) Spatial complexity in D_3-RF

(j) Accuracy in D_4-RF (k) Time in D_4-RF (l) Spatial complexity in D_4-RF

(m) Accuracy in D_5-RF (n) Time in D_5-RF (o) Spatial complexity in D_5-RF

Figure 1: Accuracy, time, and spatial complexity of RF.



(a) Accuracy in D_1-GB (b) Time in D_1-GB (c) Spatial complexity in D_1-GB

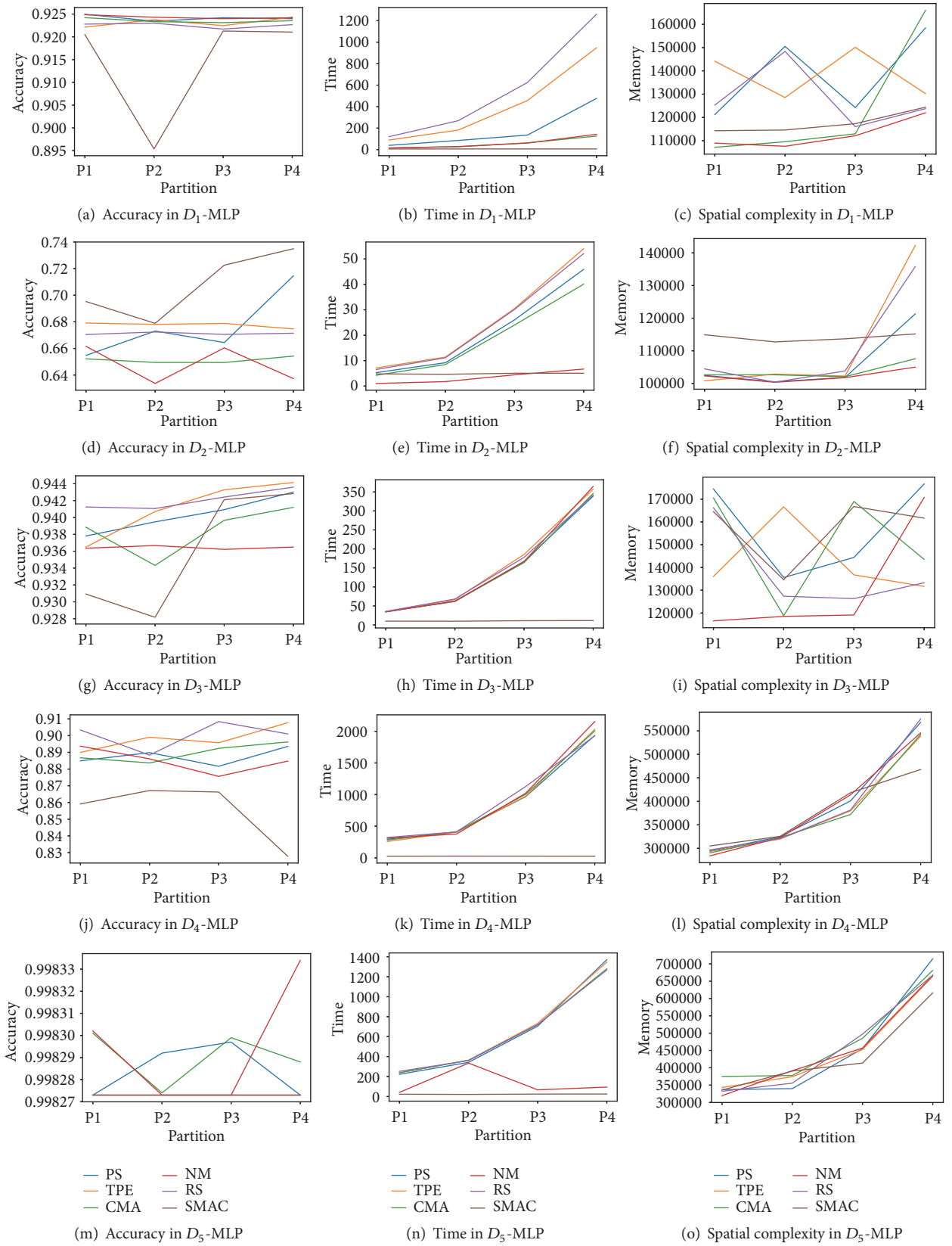
(d) Accuracy in D_2-GB (e) Time in D_2-GB (f) Spatial complexity in D_2-GB

(g) Accuracy in D_3-GB (h) Time in D_3-GB (i) Spatial complexity in D_3-GB

(j) Accuracy in D_4-GB (k) Time in D_4-GB (l) Spatial complexity in D_4-GB

(m) Accuracy in D_5-GB (n) Time in D_5-GB (o) Spatial complexity in D_5-GB

Figure 2: Accuracy, time, and spatial complexity of GB.



(a) Accuracy in D_1-MLP (b) Time in D_1-MLP (c) Spatial complexity in D_1-MLP

(d) Accuracy in D_2-MLP (e) Time in D_2-MLP (f) Spatial complexity in D_2-MLP

(g) Accuracy in D_3-MLP (h) Time in D_3-MLP (i) Spatial complexity in D_3-MLP

(j) Accuracy in D_4-MLP (k) Time in D_4-MLP (l) Spatial complexity in D_4-MLP

(m) Accuracy in D_5-MLP (n) Time in D_5-MLP (o) Spatial complexity in D_5-MLP

Figure 3: Accuracy, time, and spatial complexity of MLP.

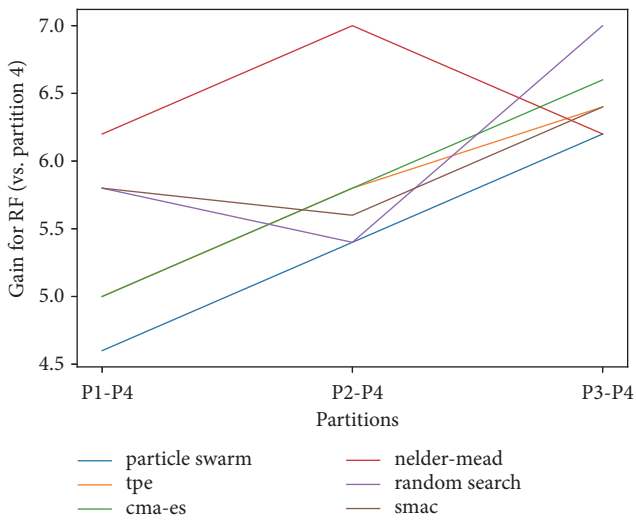


Figure 4: Average of gain for each smaller partition with respect to the whole dataset in RF.

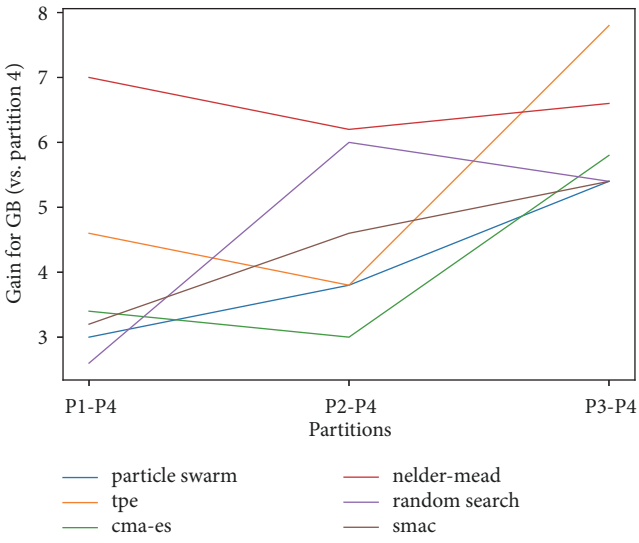


Figure 5: Average of gain for each smaller partition with respect to the whole dataset in GB.

resources and avoid loosing quality in an eventual solution based on ML.

Once we have statistically and globally analyzed the efficiency of a ML algorithm when we use smaller partitions instead of the whole dataset, the next step is going in depth over those cases in which differences are encountered. It should be noted that these could provide a gain or a loss of effectiveness. Also, a statistically difference in the accuracy, for example, could be due to a variation as low as a few ten thousandth of the total. In these cases, the relevance of this difference is meaningfully related to the order of gain or loss in the time and spatial complexity. The global average of the level of gain, according to Table 4, is included in Table 10 (see Figures 4, 5, 6, and 7).

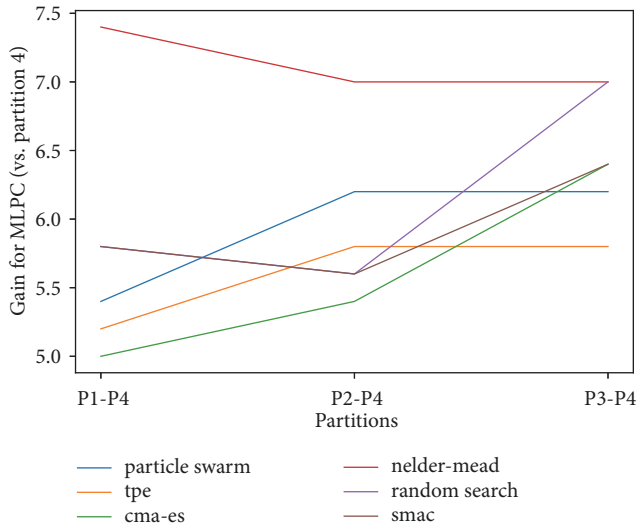


Figure 6: Average of gain for each smaller partition with respect to the whole dataset in MLP.

Table 10: Average level of profit for all ML algorithms considered and for each comparison between $P_j(D_i)$ and $P_4(D_i)$ across all HPO and D_i (%).

Combination	HPO	D_i
P_1 Vs P_4	5.04	5.04
P_2 Vs P_4	5.44	5.475
P_3 Vs P_4	6.33	6.175

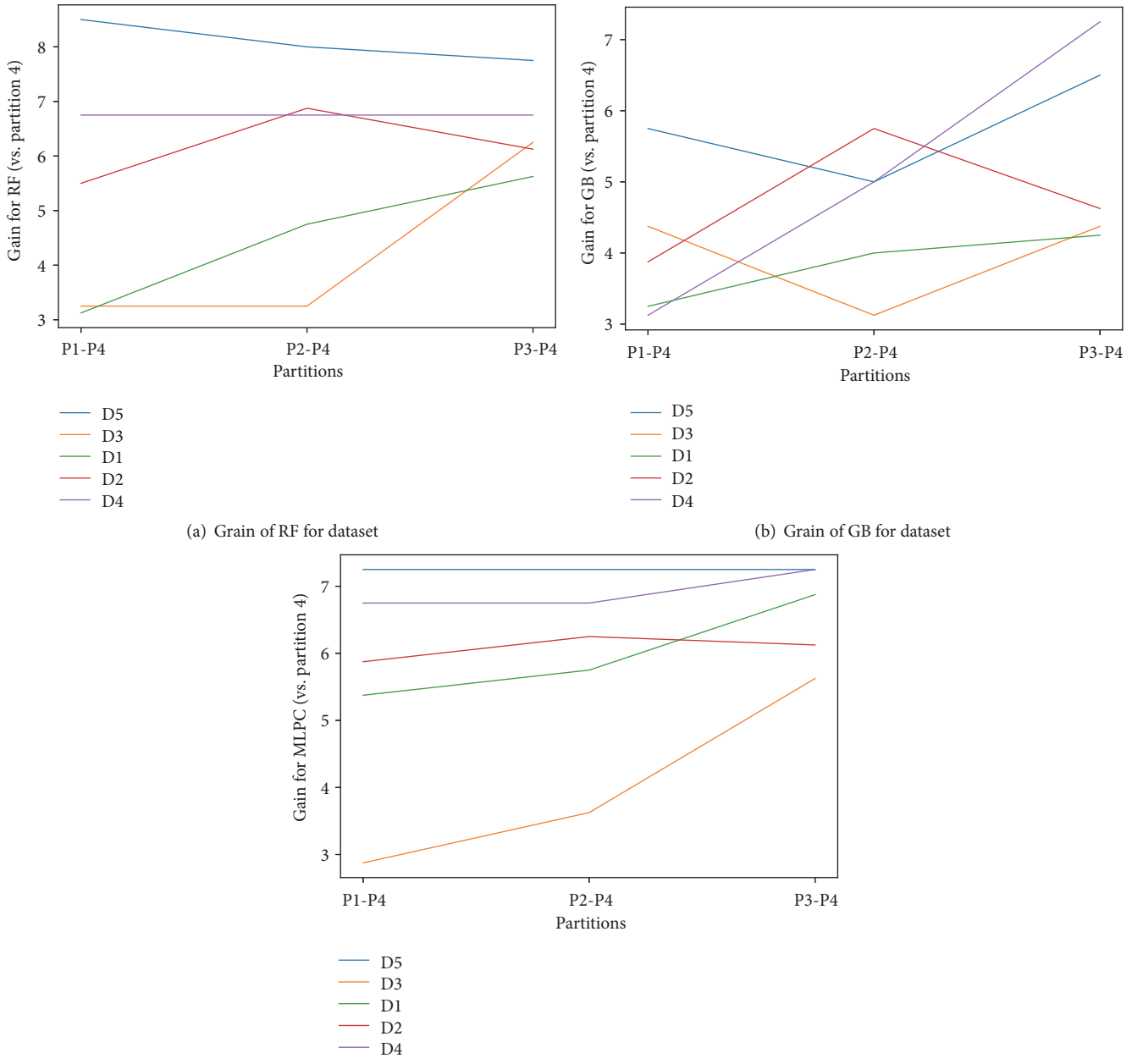
The obtained global results show an average level of profit between 5.04 and 6.33 over 9 with an increasing trend related to the size of the partition.

5. Conclusions and Future Work

Cybersecurity is a dynamical and emerging research discipline that faces on problems which are increasingly complex and that requires innovative solutions. The value of a database of Cybersecurity is very high due to the actionable knowledge that we extract from it, but in the most cases, we have to deal with a big volume of data that entails expensive costs of resources and time. Artificial Intelligence techniques, such as Machine Learning, are powerful tools that allow us to extract and generate knowledge in Cybersecurity, among other fields.

One of the main issues, in order to reach quality results by Machine Learning, is the optimization of the hyperparameter values of the algorithm. However, the automatic HPO methods suppose a cost in terms of dynamical complexity.

In this work, we have developed a statistical analysis of the fact of using smaller samples of the dataset for this process, and its influence on the effectiveness of the Machine Learning solution. The study was carried out over five different public datasets of Cybersecurity. The results let us conclude that working with smaller partitions turns out to be more efficient than performing the same process with the whole dataset. The obtained gain is different depending on the ML algorithm and



(a) Grain of RF for dataset

(b) Grain of GB for dataset

(c) Grain of MLP for dataset

FIGURE 7: Average of gain for each smaller partition with respect to the whole dataset in each dataset D_i.

the HPO technique providing the highest level of profit with the 50% of the dataset.

As future work, the following landmark would be to search what is the optimal partition to obtain the best gain, as well as studying other HPO methods over more types of ML algorithms.

Acronyms

CMA-ES: Covariance Matrix Adaptation EvolutionaryStrategies
GP: Gaussian Process
GB: Gradient Boosting
HPO: Hyperparameters Optimization
ML: Machine Learning
MLP: Multilayer Perceptron
NM: Nelder-Mead
PS: Particle Swarm
RBF: Radial Basis Function
RS: Random Search
RF: Random Forest
SMAC: Sequential Model Automatic Configuration
SMBO: Sequential Model-Based Optimization
SH: Successive Halving
TPE: Tree Parzen Estimators.

Data Availability

The datasets supporting this meta-analysis are from previously reported studies and datasets, which have been cited. The processed data are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors would like to thank the Spanish National Cybersecurity Institute (INCIBE), who partially supported this work. Also, in this research, the resources of the Centro de Supercomputación de Castilla y León (SCAYLE, www .scayle.es), funded by the "European Regional Development Fund (ERDF)", have been used.

References

[1] Z. Cheng and Z. Lu, "A Novel Efficient Feature Dimensionality Reduction Method and Its Application in Engineering," *Complexity*, vol. 2018, Article ID 2879640, 14 pages, 2018.

[2] I. Czarnowski and P. Jędrzejowicz, "An Approach to Data Reduction for Learning from Big Datasets: Integrating Stacking, Rotation, and Agent Population Learning Techniques," *Complexity*, vol. 2018, Article ID 7404627, 13 pages, 2018.

[3] F. Provost, D. Jensen, and T. Oates, "Efficient progressive sampling," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 23–32, San Diego, Calif, USA, 1999.

[4] W. Ng and M. Dash, "An Evaluation of Progressive Sampling for Imbalanced Data Sets," in *Proceedings of the Sixth IEEE International Conference on Data Mining Workshops*, Hong Kong, China, 2006.

[5] F. Portet, F. Gao, J. Hunter, and R. Quiniou, "Reduction of Large Training Set by Guided Progressive Sampling: Application to Neonatal Intensive Care Data," in *Proceedings of the Intelligent Data International Workshop on Analysis in Medicine and Pharmacology (IDAMAP2007)*, pp. 1-2, Amsterdam, Netherlands, July 2007.

[6] G. M. Weiss and F. Provost, "Learning when training data are costly: The effect of class distribution on tree induction," *Journal of Artificial Intelligence Research*, vol. 19, pp. 315–354, 2003.

[7] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2013)*, pp. 847–855, 2013.

[8] Y. Bengio, "Gradient-based optimization of hyperparameters," *Neural Computation*, vol. 12, no. 8, pp. 1889–1900, 2000.

[9] J. Luketina, M. Berglund, K. Greff, and T. Raiko, "Scalable gradient-based tuning of continuous regularization hyperparameters," *CoRR*, 2015, https://arxiv.org/abs/1511.06727.

[10] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning (ICML'15)*, vol. 37, pp. 2113–2122, 2015, http://dl.acm.org/citation.cfm?id=3045118.3045343.

[11] L. Mockus, V. Tiesis, and A. Zilinskas, "The application of bayesian methods for seeking the extremum," *Towards Global Optimization*, 1978.

[12] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to derivatibe-free otpimization, MPS/SIAM Series on Optimization*, 2009, http://epubs.siam.org/doi/book/10.1137/1.9780898718768.

[13] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[14] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[15] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[16] X. C. Guo, J. H. Yang, C. G. Wu, C. Y. Wang, and Y. C. Liang, "A novel LS-SVMs hyper-parameter selection based on particle swarm optimization," *Neurocomputing*, vol. 71, no. 16-18, pp. 3211–3215, 2008.

[17] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*, Morgan Kaufmann, 2001.

[18] F. Fortin, F. De Rainville, M. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, 2012, https://github.com/DEAP/deap.

[19] E. Hazan, A. Klivans, and Y. Yuan, "Hyperparameter Optimization: A Spectral Approach," in *In Proceedings of the Sixth International Conference on Learning Representations*, 2018.

[20] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proceedings of the Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012 (NIPS 2012)*, pp. 2960–2968, 2012.

[21] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proceedings of the Neural Information Processing Systems 2011 (NIPS 2011)*, pp. 2546–2554, 2011.

[22] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential Model-Based Optimization for General Algorithm Configuration," in *Learning and Intelligent Optimization (LION)*, C. A. C. Coello, Ed., vol. 6683 of *Lecture Notes in Computer Science*, pp. 507–523, Springer, Berlin, Germany, 2011.

[23] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[24] W. Konen, P. Koch, O. Flasch et al., "Tuned Data Mining: A Benchmark Study on Different Tuners," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO-2011)*, SIGEVO/ACM, New York, NY, USA, 2011.

[25] E. R. Sparks, A. Talwalkar, D. Haas et al., "Automating model search for large scale machine learning," in *Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC 2015)*, pp. 368–380, 2015.

[26] I. Ilievski, T. Akhtar, J. Feng, and C. A. Shoemaker, "Efficient hyperparameter optimization for deep learning algorithms using deterministic RBF surrogates," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pp. 822–829, San Francisco, Calif, USA, 2017.

[27] Y. Ozaki, M. Yano, and M. Onishi, "Effective hyperparameter optimization using Nelder-Mead method in deep learning," in *Proceedings of the IPS Transactions on Computer Vision and Applications*, pp. 9–20, 2017.

[28] K. G. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *In Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS 2016)*, pp. 240–248, 2016.

[29] B. Recht, *Embracing the random*, 2016, http://www.argmin.net/2016/06/23/hyperband.

[30] B. Recht, *The news of auto-tuning*, 2016, http://www.argmin.net/2016/06/20/hypertuning/.

[31] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: a novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research (JMLR)*, vol. 18, pp. 1–52, 2017.

[32] P. Koch, B. Wujek, O. Golovidov, and S. Gardner, "Automated Hyperparameter Tuning for Effective Machine Learning," in *Proceedings of the SAS Global Forum 2017 Conference*, SAS Institute Inc, Cary, NC, USA, 2017, http://support.sas.com/resources/papers/proceedings17/SAS514-2017.pdf.

[33] K. Eggensperger, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Efficient benchmarking of hyperparameter optimizers via surrogates," in *Proceedings of the Twenty-Ninth Association for the Advancement of Artificial Intelligence*, pp. 1114–1120, 2015.

[34] P. Turney, "Types of cost in inductive concept learning," in *Proceedings of the ICML '2000 Workshop on Cost-Sensitive Learning*, pp. 15–21, 2000.

[35] D. Bogdanova, P. Rosso, and T. Solorio, "Exploring high-level features for detecting cyberpedophilia," *Computer Speech and Language*, vol. 28, no. 1, pp. 108–120, 2014.

[36] P. Galán-García, J. Gaviria de la Puerta, C. Laorden Gómez, I. Santos, and P. G. Bringas, "Supervised Machine Learning for the Detection of Troll Profiles in Twitter Social Network: Application to a Real Case of Cyberbullying," in *Proceedings of the International Joint Conference SOCO'13-CISIS'13-ICEUTE'13. Advances in Intelligent Systems and Computing*, Á. Herrero et al., Ed., vol. 239, Springer, 2014.

[37] S. Miller and C. Busby-Earle, "The role of machine learning in botnet detection," in *Proceedings of the 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pp. 359–364, Barcelona, Spain, 2016.

[38] J. R. Moya, N. DeCastro-García, R. Fernández-Díaz, and J. L. Tamargo, "Expert knowledge and data analysis for detecting advanced persistent threats," *Open Mathematics*, vol. 15, no. 1, pp. 1108–1122, 2017.

[39] A. Shenfield, D. Day, and A. Ayesh, "Intelligent intrusion detection systems using artificial neural networks," *ICT Express*, vol. 4, no. 2, pp. 95–99, 2018.

[40] F. Vanhoenshoven, G. Napoles, R. Falcon, K. Vanhoof, and M. Koppen, "Detecting malicious URLs using machine learning techniques," in *Proceedings of the 2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, Athens, Greece, December 2016.

[41] M. Zanin, M. Romance, S. Moral, and R. Criado, "Credit Card Fraud Detection through Parenclitic Network Analysis," *Complexity*, vol. 2018, Article ID 5764370, 9 pages, 2018.

[42] N. DeCastro-García, Á. L. Muñoz Castañeda, M. Fernández Rodríguez, and M. V. Carriegos, "On Detecting and Removing Superficial Redundancy in Vector Databases," *Mathematical Problems in Engineering*, vol. 2018, Article ID 3702808, 14 pages, 2018.

[43] V. N. Gudivada, R. Baeza-Yates, and V. V. Raghavan, "Big data: Promises and problems," *The Computer Journal*, vol. 48, no. 3, pp. 20–23, 2015.

[44] R. Baeza-Yates, "Big Data or Right Data," in *Proceedings of the 7th Alberto Mendelzon International Workshop on Foundations of Data Managements, CEUR Workshop Proceedings 1087*, p. 14, 2013.

[45] S. Garcia, A. Fernandez, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power," *Information Sciences*, vol. 180, no. 10, pp. 2044–2064, 2010.

[46] F. Pedregosa, G. Varoquaux, A. Gramfort et al., "Scikit-learn: machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[47] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[48] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Books, Washington, DC, USA, 1961.

[49] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D. E. Rumelhart, J. L. McClelland, and The PDP research group, Eds., vol. 1, MIT Press, 1986.

[50] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, https://arxiv.org/abs/1412.6980.

[51] J. Bergstra J, D. Yamins, and D. Cox, "Hyperopt: a Python library for optimizing the hyperparameters of machine learning algorithms," in *In Proceedings of the 12th Python in Science Conference (SciPy 2013)*, pp. 13–20, 2013.

[52] M. Claesen, J. Simm, D. Popovic, Y. Moreau, and B. De Moor, *Easy Hyperparameter Search Using Optunity*, 2014, https://github.com/claesenm/optunity.

[53] M. Lindauer, K. Eggensperger, M. Feurer et al., *SMAC v3: Algorithm Configuration in Python*, 2017, https://github.com/automl/SMAC3.

[54] M. Hopkins, E. Reeber, G. Forman, and J. Suermondt, "Dataset Spambase," UCI Machine Learning Repository [http://archive.ics.uci.edu/ml/datasets/spambase]. Irvine, CA: University of California, School of Information and Computer Science. 2017.

[55] Á. M. Guerrero-Higueras, N. DeCastro-García, and V. Matellán, "Detection of Cyber-attacks to indoor real time localization systems for autonomous robots," *Robotics and Autonomous Systems*, vol. 99, pp. 75–83, 2018.

[56] M. Rami, T. L. McCluskey, and F. A. Thabtah, "An Assessment of Features Related to Phishing Websites using an Automated Technique," in *Proceedings of the International Conferece For Internet Technology And Secured Transactions (ICITST 2012)*, pp. 492–497, IEEE, London, UK, 2012.

[57] R. M. Mohammad, F. Thabtah, and L. McCluskey, "Intelligent rule-based phishing websites classification," *IET Information Security*, vol. 8, no. 3, pp. 153–160, 2014.

[58] R. M. Mohammad, F. Thabtah, and L. Mc-Cluskey, "Predicting phishing websites based on self-structuring neural network," *Neural Computing and Applications*, vol. 25, no. 2, pp. 443–458, 2014.

[59] L. Dhanabal and S. P. Shantharajah, "A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 6, 2015.

[60] "NSL-KDD Dataset," https://github.com/defcom17/NSL_KDD, 2015.

[61] A. D. Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi, "Calibrating Probability with Undersampling for Unbalanced Classification," in *Symposium on Computational Intelligence and Data Mining (CIDM)*, IEEE, 2015, https://www.openml.org/d/1597.