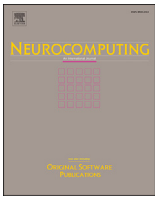
Neurocomputing 670 (2026) 132508

Contents lists available at ScienceDirect

Neurocomputing

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



Bayesian generation of synthetic datasets for machine-learning tasks: a performance study

Paolo Fosci [a,*], Javier Nieves [b], Giuseppe Psaila [a], Jacopo Boffelli [a], Pablo Garcia Bringas [c]

[a] University of Bergamo, Dept. of Management, Information and Production Engineering, Viale Marconi 5, Dalmine, 24044, (BG), Italy
[b] Azterlan Member of Basque Research Team Alliance (BRTA), Aliendalde Etxetaldea 6, Durango, 48200, (BI), Spain
[c] University of Deusto, Faculty of Engineering, Unibertsitate Etorb. 24, Bilbao, 48007, (BI), Spain

ARTICLE INFO

Communicated by L. Ma

Keywords:
Generation of synthetic data
Bayesian generation
Bayesian networks
The *YABaGen* tool
Effectiveness and efficiency

ABSTRACT

Performing Machine Learning (ML) tasks on large-scale datasets, as well as simply storing them for subsequent analysis or for long-term archival, require large computational power. The described approach builds on the technique known as "Bayesian Generation" to produce synthetic datasets in such a way that the probability distribution in the source dataset is maintained as much as possible in the new synthetic ones, even if they are much smaller than the original (large) dataset. In fact, this study investigates the impact of generating smaller synthetic datasets for training ML models in place of the original dataset, adopting a twofold perspective. Firstly, the impact on the effectiveness of ML models trained on these smaller synthetic datasets is assessed. Secondly, the amount of computational resources required to generate the synthetic datasets, train ML models on them, and perform the testing phase is measured. Specifically, both execution time and main memory usage are taken into account. Finally, this research work shows that the loss in terms of effectiveness remains consistently limited and stable, and it identifies the scenarios and ML techniques for which incorporating the generation of small synthetic datasets into the ML pipeline can be beneficial for practical deployment in environments with constrained computational resources, such as mobile or industrial devices.

1. Introduction

The term "digital transition" is the keyword that characterizes the currently ongoing transition toward a fully digital world, in which almost every activity is digitally assisted. Currently, one widely used buzzword is "digital twin" [1], referring to the creation of a virtual counterpart of a physical device that enables the simulation and analysis of the behavior of the real-world device. This concept is gaining increasing relevance in industrial environments [2], as it promises to enhance the efficiency of production processes. In particular, within these environments, production machinery is becoming more and more digital, driven by the Industry 4.0 paradigm and the Industrial Internet of Things (IIoT). As a result, it is possible to collect an incredible amount of data related to manufacturing operations, which hopefully can later be exploited for data mining and other Machine Learning (ML) tasks [3].

Nonetheless, the digital transition is occurring across all human activity. Consequently, incredibly large datasets are continuously generated and stored. In that way, exploiting these datasets to learn about human processes could be a crucial activity, enabling the extraction of meaningful value from the collected data. This perspective underlies Big Data and the so-called "7-V Model" [4].

Typically, when ML models are trained on large datasets, a major challenge arises. The challenge lies in the substantial amount of computational resources required for performing the training phase. Equally, a second challenge concerns data storage. It means the problem of maintaining these large datasets over extended periods. This leads to high associated costs. Hence, if a compact yet approximately equivalent representation of the data could be preserved instead, storage requirements and the overall costs would be significantly reduced.

To address the aforementioned issues, in [5] the authors presented a novel technique for generating synthetic datasets, named *Bayesian Generation*. The described approach is based on "Bayesian Networks" with the purpose of extracting the probability distribution of values in the source dataset.

The technique is implemented within the *YABaGen* tool, which can be used as follows: (i) given a large source dataset D, *YABaGen* takes

* Corresponding author.
Email address: paolo.fosci@unibg.it (P. Fosci).

https://doi.org/10.1016/j.neucom.2025.132508
Received 7 June 2025; Received in revised form 29 November 2025; Accepted 19 December 2025
Available online 24 December 2025

it as input and generates a novel dataset S such that the probability distribution of S closely approximates the probability distribution of D; (ii) if the size of S is smaller than the size of D, its storage costs are reduced, and training data-driven models on S should also require fewer computational resources than training on D.

In this scenario, several research questions emerge regarding the potential practical exploitation of *YABaGen*. The first question is: (Q1) Is there a (significant) loss when ML models are trained on a small synthetic dataset in place of the source dataset?

The second question is: (Q2) Is there any practical advantage in terms of necessary computational resources during the training phase and/or during the test phase, if the ML models are trained on the small synthetic dataset?

And finally: (Q3) Are the computational resources necessary to generate the small synthetic dataset justified in the overall process?

In [5], only a preliminary evaluation of effectiveness was performed. The novel contribution of this paper is to answer the aforementioned research questions, by conducting a performance study on real-world datasets. During the study, several popular ML techniques were considered. Specifically, all selected models were trained either on the source datasets or on a variety of synthetic datasets generated by *YABaGen* with progressively reduced sizes.

Three different aspects are evaluated: (i) the effectiveness in terms of accuracy between the models trained on the source datasets and the models trained on the synthetic ones; (ii) the execution time required to generate the synthetic datasets, as well as the execution time needed to perform both the training and testing phases of each selected ML technique, evaluated on the original dataset and on each synthetically generated dataset; (iii) the amount of main (RAM) memory required for both the training and testing phases, evaluated on both the original and synthetic datasets.

Regarding the performance study, the main goals are the following: (i) showing the effectiveness of the *Bayesian Generation* technique in terms of a very small loss of accuracy for ML techniques that are trained on synthetic datasets that are smaller than the source dataset; (ii) understanding which ML techniques actually consume large amounts of computational resources; and (iii) understanding the impact of generating synthetic datasets through *Bayesian Generation* in the context of ML applied to large datasets.

A further contribution of the paper is understanding how *Bayesian Generation* and *YABaGen* are positioned with respect to state-of-the-art techniques for synthetic data generation. The paper will show that both in terms of the quality of generated datasets and the stability of performance of trained ML models, *YABaGen* is fully comparable. Furthermore, the paper will show that *YABaGen* is faster while generating small synthetic datasets.

The remainder of the paper is organized as follows. Section 2 provides a short overview of techniques for synthetic-data generation. Section 3 presents the technique for *Bayesian Generation* that was devised by the authors and implemented within the *YABaGen* tool. Section 4 presents how the experimental campaign was conducted, in preparation for the performance study. After that, Section 5 analyzes the loss of accuracy when ML models are trained on (small) synthetic datasets. Then, Section 6 analyzes the consumption of computational resources, i.e., execution time and occupied main memory during the various phases. Section 7 discusses the results in relation to the three above-presented research questions. Finally, Section 8 draws the conclusions and sketches possible future work.

2. Related work

The work on Synthetic-Data Generation (SDG) has emerged as a powerful paradigm across diverse scientific domains, from molecular simulation and astrophysics to machine learning, privacy-preserving analytics, and theoretical learning frameworks. Specifically, this section aims to summarize representative works in the aforementioned fields. In fact, these works are organized into three main thematic groups: (i) simulation-based synthetic-data generation; (ii) machine learning-based models; and (iii) theoretical perspectives focused on learnability and compression.

2.1. Simulation-based synthetic data

Early approaches to SDG originated in the field of natural and physical sciences, where simulation techniques such as Monte Carlo methods were employed to model phenomena that could not be directly observed; as an example, the reader can think of molecular dynamics as well as astrophysical modeling [6,7]. These methods prioritized physical realism and adherence to domain-specific constraints, so as to produce synthetic observations that are consistent with underlying physical laws. While valuable within their respective domains, these methods differ fundamentally from this work, which focuses on data-driven and probabilistic modeling to optimize machine learning performance and computational efficiency.

2.2. Machine learning-based generative models

Modern SDG has increasingly moved toward machine learning-driven paradigms, focusing on statistical fidelity, privacy preservation, and scalability across diverse data types. Comprehensive evaluations, such as [8], compare classical oversampling methods like SMOTE [9] and non-parametric approaches such as synth-pop [10] with neural generative models including GANs [11], VAEs [12], and CTGAN [13]. These studies reveal a key trade-off between model complexity and computational efficiency. In fact, simpler models better preserve basic statistical relationships, whereas deep generative architectures provide higher flexibility but demand significantly greater computational resources. This balance between accuracy, scalability, and resource cost remains one of the central challenges in SDG.

Recent surveys [14] further highlight the rapid diversification of SDG models, from GAN-based and diffusion architectures [15] to transformer-based generators for sequential or high-fidelity data. Extensions such as DP-GANs for privacy [13] and domain-specific virtual environments [16] illustrate the growing specialization of generative models. Other lines of work explore reinforcement learning-based generation, as in SeqGAN [17], to adapt synthetic data creation to structured or goal-driven tasks. While these approaches offer expressive and powerful mechanisms for data synthesis, they often suffer from high computational demands and limited interpretability. This motivates the need for lighter probabilistic alternatives, such as Bayesian-based SDG, that maintain statistical coherence while enabling efficient deployment in constrained or embedded computing environments.

2.3. Theoretical foundations: learnability and compression

In addition to the aforementioned works focused on empirical and simulation-based frameworks, foundational theories in computational learning provide a formal background for understanding the feasibility and limitations of synthetic data generation. In particular, studies on sample compression and PAC-learnability (e.g., [18–20]) establish that, if a concept class can be represented through a compact compression scheme, then it remains learnable within the Probably Approximately Correct (PAC) framework. This connection between data compression, representation, and generalization illustrates how synthetic data, when generated under probabilistic constraints, can preserve the essential statistical properties of the original distribution while reducing its complexity.

Subsequent developments have extended these theoretical principles to structured and high-dimensional data, demonstrating that compressed or reconstructed representations can still enable effective learning. Although these works do not address synthetic data generation directly, they delineate the theoretical boundaries within which SDG can be considered meaningful and provably effective. From this

perspective, the act of generating synthetic data can be viewed as producing a compressed approximation of the underlying data-generating process, something that conceptually aligns with the probabilistic and Bayesian modeling approach adopted in the present work.

2.4. Positioning and contribution

This paper introduces an approach focused on a Bayesian network-based technique [5] for probabilistic synthetic data generation, aimed at preserving the joint probability distribution of the source dataset while reducing its size. Unlike prior studies mainly focused on privacy [13], data augmentation [9], or domain-specific simulations [16], the proposed approach investigates how Bayesian-based SDG can decrease the computational footprint of machine learning pipelines. Specifically, it analyzes (i) the effectiveness of models trained on compact synthetic datasets and (ii) the computational resources required for SDG, training, and testing in terms of time and memory (see Section 4).

In contrast to large-scale generative models such as diffusion models [15], the proposed approach targets resource-constrained environments (e.g., edge or industrial devices), allowing for efficient data-driven solutions with minimal loss of accuracy. Overall, this work positions Bayesian-based SDG as a practical tool for optimizing ML workflows, focusing on cost-efficiency, scalability, and real-world usability rather than generative realism.

3. Bayesian generation

This section presents and describes the method for *Bayesian Generation*. Specifically, the method is implemented within the *YABaGen* tool (where the acronym stands "Yet Another Bayesian Generator"). *YABaGen* is programmed in Java, so that it can directly utilize components provided within the WEKA suite [21], the well-known open-source software written in Java, which offers a collection of tools and algorithms for machine learning.

The operations performed by the tool are summarized below.

- The input dataset, named $Source$, is a tabular dataset, where records have all the same columns (or variables, in the following) $(a_1, \ldots, a_i, \ldots, a_n)$. Variables may be either categorical or numerical. The last variable a_n is called the *class* variable and is necessarily categorical. In the following, $D(a_i)$ denotes the domain of the variable a_i in the dataset $Source$, i.e., the set of values of a_i that appear in $Source$.
- The symbol *gf* denotes the "generation factor", in other words, the size of the output for the new $Synthetic$ dataset. This factor is expressed as a percentage of $Source$. Formally, it is:
$$|Synthetic| = |Source| \times gf/100.$$
- The output dataset, $Synthetic$, is generated such that its records have the same structure as those in $Source$, i.e., $(a_1, \ldots, a_i, \ldots, a_n)$; as records in $Source$, each variable a_i still has the same domain $D(a_i)$. The generated records aim to preserve, as closely as possible, the probability distribution underlying $Source$. As a consequence, it is possible that $Source \cap Synthetic$ is empty.

The remainder of this section presents the method for *Bayesian Generation* and its steps in detail. In addition, Fig. 1 complements the explanation, showing a graphical summary of the process.

3.1. Preprocessing the source dataset

First of all, the source dataset is preprocessed, in order to properly manage the numerical values of variables. In particular, when a variable a_i is numeric, the generated synthetic dataset is expected to retain this numerical nature. Moreover, the values of a_i in $Synthetic$ typically fall within $D(a_i)$, i.e., the set of values actually observed in $Source$. Thus, $Source$ is preprocessed, as described hereafter.

- A "Discretization Algorithm" identifies the set $C(a_i)$ of categorical values (states) c_k, for each numerical variable a_i.
- Then, for each numerical variable a_i and for each categorical value $c_k \in C(a_i)$, the set $N(a_i, c_k)$ of bi-tuples(n_z, p_z) (where p_z is the number of occurrences of the value n_z, thus, p_k implicitly represents the probability of n_k) is computed, by scanning $Source$ again.
- Contextually, the intermediate version, $Categorical$, of the dataset $Source$ is generated by replacing each numerical value$n_z \in D(a_i)$ with the corresponding categorical value c_k.

3.2. Learning the Bayesian network

This step obtains the inherent probability distribution in the source dataset. A "Bayesian Classifier" (such as the WEKA tool named "Tree Augmented Naïve" (or *TAN*) trains a model from the dataset $Categorical$. The learned model, a specific Bayesian Network, represents the approximate probability distribution of the way each variable influences the other variables. In order to be effective, only categorical values must be considered for variable states, to avoid dispersion of probabilities on a continuous domain; this explains why $Source$ is transformed into $Categorical$ by the already explained preprocessing step.

A Bayesian classifier builds the Bayesian network in two phases.

1. **Structure Identification.** First of all, the structure of the network is learned; in other words, the relationships between variables. Hence, each node of the network corresponds to a variable from the dataset and each arrow represents a dependent relationship between two variables.
2. **Computation of Probability Distributions.** Then, the strength of relationships between variables and their corresponding values is estimated. To this end, using the network structure identified during the first phase, these dependencies are encoded as "probability tables" associated with each node of the network.

Among the various algorithms that could be employed (such as the *Hill Climber* algorithm [22] or the *K2* algorithm [23]), the TAN algorithm [23] was selected following a preliminary experimental assessment. Although both *K2* and *Hill Climber* were also evaluated, neither showed satisfactory results in our setting.

3.3. Sorting nodes

In order to generate a new record during the final step of this data generation (see Section 3.5 for further details) the values of all variables must be sampled according to the probability distributions specified by the Bayesian Network. When traversing the nodes of the network, these probabilities vary according to the conditional dependencies imposed by parent nodes. Consequently, the nodes must be processed following a topological ordering.

A topological sorting algorithm (omitted here for brevity and widely explained in [5]) produces an ordered sequence of nodes $\mathfrak{N} = \{a_1, \ldots, a_n\}$. The first node in the sequence, a_1, is always the class variable *cl* to be predicted (see the structure of $Source$ at the beginning of this section). The remaining variables follow the order such that for any node a_i, if the corresponding variable has parents, all of them appear in the prefix formed by $[a_1, a_{i-1}]$. Likewise, all of its children, that is, the variables that depend on it, appear in the suffix $[a_{i+1}, a_n]$.

Thus, when generating the state for the variable a_{i+1}, the states of all its parent variables will already have been assigned. Consequently, it is possible to directly access the corresponding entry in its conditional probability table.

3.4. Representing the probability distribution

The sequence of sorted variables $\mathfrak{N}$ will be followed to generate the corresponding variable states (see Section 3.5). Therefore, it is necessary to extract the probability distribution encoded in the Bayesian Network and represent it in a suitable form for this generation process.

A list of "variable descriptors" $\mathcal{VD} = \{vd_1, \ldots, vd_n\}$ is built, such that vd_i (with $1 \leq i \leq n$ and $n = |\mathfrak{N}|$) corresponds to the variable $\mathfrak{N}[i]$.

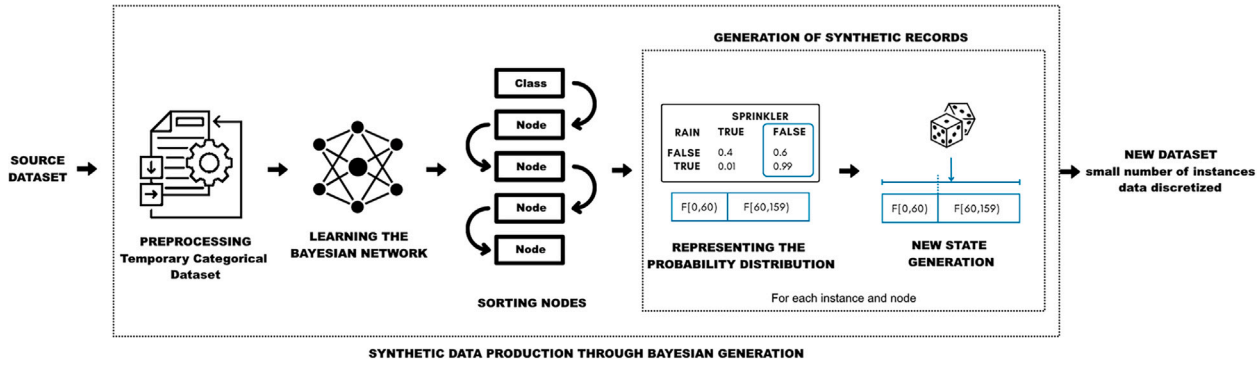


Fig. 1. Overview of the method for generating synthetic data.

The goal of each variable descriptor vd_i is to represent the conditional probability of a state for $\mathfrak{N}[i]$, based on the states chosen for the "parent variables" (i.e., variables $\mathfrak{N}[j]$ with index $1 \leq j \leq i$). Remember that, depending on the states chosen for parent variables, the probabilities of the states for a_i vary.

Each variable descriptor vd_i contains a "probability table" with three entries: (i) the sequence of states of the parents; (ii) a state of the current variable a_i; (iii) the conditional probability of the state, given the parents' states. As a practical result, the probability table can be queried by providing the sequence of parents' states.

By querying the probability table based on a sequence of parent states $path_{i-1}$, a set of bi-tuples of the form (c_j, p_j) is extracted, where c_j refers to a variable state and p_j refers to the probability associated with that state ($0 \leq p_j \leq 1$). This set of bi-tuples describes the probability distribution to be used for generating a variable state, because it is based on the given conditional probabilities stored in the Bayesian Network.

In order to simplify the exploitation of a variable descriptor in the final step (see Section 3.5), all probabilities are rounded to the second fractional digit and then multiplied by 100. This way, the value p_j actually stored in a bi-tuple is an integer number. Thus, probabilities are represented in a linear way from the value 1 to S (assuming that states c_j with $p_j = 0$ are not considered), where S is the sum of the integer numbers associated with each state. This range will be divided into different blocks of unequal size that mark the sections associated with each of the possible states for the variable described by vd_i.

3.5. Generation of a new record

The final step performed by our synthetic data generator is the production of new instances (records) in the dataset $Synthetic$.

For each variable in the sorted sequence $\mathfrak{N}$, a value (state) is generated in order to construct a new synthetic record. The procedure is carried out as follows.

- For the class variable $\mathfrak{N}[1]$, the state v_1 is generated by applying the Monte Carlo method (see later).
  A sequence $seq_1 = (v_1)$ is initialized.
- For each variable $\mathfrak{N}[i]$, in position $1 < i$, its probability table is queried by the sequence seq_{i-1}, which reports the values generated for the parent variables. The result of the query is the set of bi-tuples (c_j, p_j) with the conditional probabilities of possible states for the current variable, provided that values reported in seq_{i-1} are generated. Again, a value v_i is generated through the Monte Carlo method and the novel sequence $seq_i = seq_{i-1} \bullet (v_i)$ is generated (where $\bullet$ is the concatenation operator).

The result is a new synthetic record. The generation process is repeated as the generation factor gf indicates, i.e., $|Synthetic| = |Source| \times gf/100$.

Pseudo-Random Generation. To generate the states of variables, the sampling theory explained in the Monte Carlo methods is used [24].

For each variable a_i, the method generates the state as follows.

- The set BT_i of bi-tuples is obtained from within the probability table in the variable descriptor $\mathcal{VD}[i]$; if $i = 1$ (class variable), BT_1 is the entire probability table in $\mathcal{VD}[1]$; otherwise, BT_i contains only the bi-tuples that are associated with the sequence of parents' states seq_{i-1}.
- For each bi-tuple $bt_j \in BT_i$, a range R_j is computed, such that $R_j = [l, u]$, with $l < u$; for $j > 1$, $R_j.l = R_{j-1}.u + 1$ and $R_j.u = R_{j-1}.u + p_j$; for $j = 1$, $R_1.l = 1$ and $R_1.u = p_1$. Note that $R_n.u = S$.
- A random number n in the range $[1, S]$ is generated and the state c_j corresponding to the range R_j such that $n \in R_j$ is generated as state (value) for the variable, i.e., $v_i = c_j$.

This way, for each value c_j, the probability $P(c_j) = (R_j.u - R_j.l + 1)/S$. Notice that by generating a large number M of trials, the distribution p_j can be approximated with an error rate of $1/\sqrt{M}$.

Generating Numerical Values. Remember that states for variables are categorical values that were obtained through the initial discretization process (see Section 3.1); also recall that each categorical value c_j of a variable a_i has an associated set $N(a_i, c_k)$ of bi-tuples (n_z, p_z). Consequently, the Monte Carlo method is used to generate a numerical value that was present in *Source* for a_i while maintaining the same distribution of values. As a result, *Synthetic* replicates the same structure as in *Source*.

4. Experimental campaign

This section provides a detailed description of the experimental workflow adopted to evaluate the effectiveness and impact of synthetic data generation using the *YABaGen* tool. The experimental campaign was carefully designed to assess how the incorporation of synthetic instances influences classification accuracy, resource utilization, and overall behavior across multiple ML algorithms.

Specifically, Fig. 2 illustrates the workflow explained below.

4.1. Hardware and software

The experiments were conducted on a virtualized IT infrastructure that is hosted on-premises at Azterlan (ES), composed of two Dell EMC PowerEdge R450 servers configured in a high-availability VMware ESXi 8.0.2 cluster.

Virtual machines were stored in a shared DellEMC PowerVault ME5012 storage cabinet connected via redundant SAS interfaces. Each host featured 32 GB RAM and supported software such as JDK 1.8/1.14 and WEKA for data-processing tasks. Additional computational resources were available through two DellEMC PowerEdge R750xa systems, each equipped with dual Intel Xeon Gold 6326 CPUs (2.90 GHz), 256 GB RAM, and 2 × 448 GB SSDs.

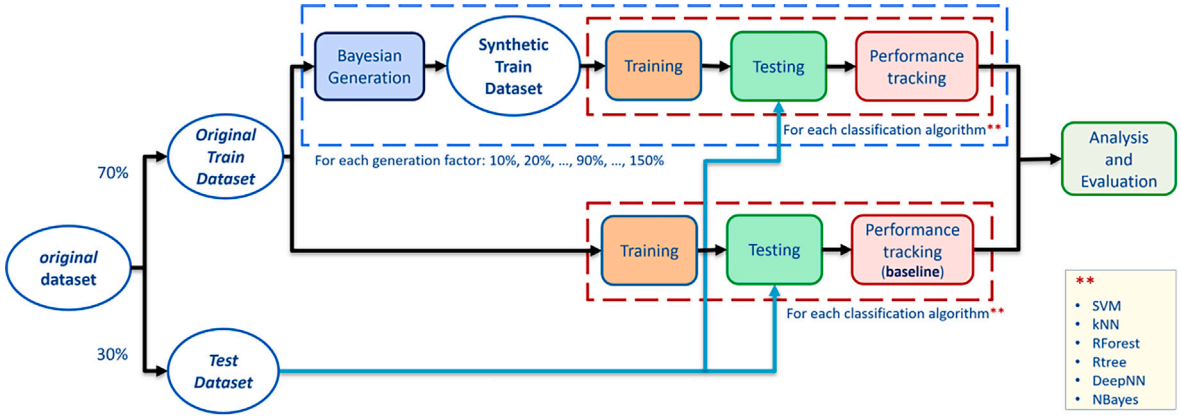


Fig. 2. Schema of the methodology exploited for the experimental campaign.

These machines also included GPUs, with 2 units allocated to virtual machines and 2 available for experimental use.

4.2. Dataset preparation

For this research, two large original datasets were used for the experimental campaign, both downloaded from Kaggle, the online community platform for data scientists and machine learning practitioners. The first dataset, hereafter referred to as $DS1$, is the "Smoking and Drinking Dataset with Body Signal".[1] The second dataset, hereafter referred to as $DS2$, is the "Cardiovascular Disease Dataset".[2]

The dataset $DS1$ comprises $991,346$ rows and, to ensure an unbiased evaluation, it was randomly partitioned into 2 subsets: $30\ \%$ designated as the Test Dataset ($297,404$ rows) and $70\ \%$ as the Training Dataset ($693,942$ rows). Similarly, the dataset $DS2$, containing a total of $70,000$ rows, was also randomly partitioned into 2 subsets, with $30\ \%$ ($21,000$ rows) assigned to the Test Dataset and $70\ \%$ ($49,000$ rows) assigned to the Training Dataset.

For both $DS1$ and $DS2$, the training subsets were employed to conduct the original experiments, serving as the baseline for comparison and to generate synthetic data for reproducing the experiments. Then, a comparative evaluation of the results was conducted (for more details, see Section 4.4).

4.3. Synthetic data generation with YABaGen

Given a training dataset, *YABaGen* was run to generate a pool of synthetic datasets, using varying generation factors, from $150\ \%$, to $10\ \%$. For each factor, a synthetic dataset was generated using the original training data from *Original Train Dataset* (OTD) to produce new *Synthetic Train Datasets*. This process aims to simulate varying degrees of data production to evaluate its impact on model generalization. In the remainder of the paper, synthetic datasets are denoted as *SDxxx*; for example, *SD010* denotes the synthetic dataset whose size is $10\ \%$ of the original training dataset.

4.4. ML workflow

Each experiment follows a standardized ML workflow consisting of three steps, hereafter described.

1. **Training Phase**: Models are trained separately using
   - (i) the Original Training Dataset (baseline) and
   - (ii) each of the Synthetic Training Datasets.
2. **Testing Phase**: The trained models are evaluated using the same test dataset, ensuring consistency in performance comparison.
3. **Performance Tracking**: during both training and testing, the following metrics are captured:
   - Prediction results:
     True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN).
   - Execution time.
   - Memory consumption.

Based on the collected prediction outcomes (TP, TN, FP, FN), the *Accuracy* and *F1-score* metrics were computed to quantitatively assess the classification performance.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$\text{F1-score} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{2}$$

4.5. Classification techniques

The machine learning models selected for the experiments shown in this paper represent a diverse and well-established set of algorithms widely adopted in the literature. The primary objective was to first verify that *YABaGen* performs effectively with these classical methods, thereby providing a solid and interpretable baseline for evaluating its applicability. In addition, the selection was driven by practical considerations, as these techniques are natively supported and easily deployable within the WEKA environment, into which the proposed tool is integrated. In order to analyze variability in learning strategies, six different classification techniques available with the WEKA suite (see Table 1) were selected. Hereafter, techniques and configurations are described.

- *SVM* [25] stands for "Support Vector Machines", implemented in the WEKA tool named SMO; the default kernel named *PolyKernel* was used.

Table 1
ML techniques exploited in the experimental campaign.

Label	Technique	Weka tool
SVM	SVM [25]	SMO
kNN	kNN [26]	IBk
RForest	Random Forest [27]	RandomForest
RTree	Random Tree [28]	RandomTree
DeepNN	MLP (ANN) [29]	MultilayerPerceptron
NBayes	Naïve Bayes [30]	NaiveBayes

[1] Smoking and Drinking Dataset: https://www.kaggle.com/datasets/sooyoungher/smoking-drinking-dataset/data, accessed on 31/10/2025.
[2] Cardiovascular Disease Dataset: https://www.kaggle.com/datasets/akshatshaw7/cardiovascular-disease-dataset, accessed on 31/10/2025.

- *kNN* [26] stands for "k-Nearest Neighbors", implemented in the WEKA tool named IBk. The experiments were conducted using the parameter $k = 1$.
- *RForest* [27] stands for "Random Forests", implemented within the homonymous WEKA tool. The experiments were conducted using 100 iterations, corresponding to the number of trees in the Random Forest.
- *RTree* [28] stands for "Random Tree", implemented within the homonymous WEKA tool.
- *DeepNN* [29] stands for "Deep Neural Network"; specifically the techniques named " Multilayer Perceptron" was used, by using the homonymous WEKA tool.
- *NBayes* [30] stands for "Naïve Bayes", implemented within the homonymous WEKA tool.

4.6. Automation and repetition

To improve reliability and account for stochastic variability, the entire process is fully automated and repeated three times for each experimental condition. This applies to both the baseline dataset and each synthetic dataset.

In each iteration: (i) Models are freshly trained, (ii) Predictions are made on the same fixed test dataset, and (iii) Performance metrics are recorded for averaging and variance analysis.

In the final stage, all tracked data are aggregated to perform a detailed comparative analysis. The performance indicators are examined in terms of classification accuracy, computational efficiency (execution time), and memory consumption.

5. Evaluation: effectiveness

The first aspect that was considered during the experimental campaign is effectiveness. The research question Q1 can be summarized as follows: *Is the loss of accuracy significant or negligible, when ML models are trained on synthetic datasets, in comparison with training conducted on the original dataset?*

Indeed, one of the goals of this research is to provide a tool that enables working with and storing synthetic versions of datasets, including potentially reduced or small-scale variants.

5.1. Dataset DS1

For dataset $DS1$, multiple synthetic versions were generated from the original dataset, using regeneration factors of 150 %, 120 %, 100 %, 90 %, 80 %, 70 %, 60 %, 50 %, 40 %, 30 %, 20 %, and 10 %.

In this way, Fig. 3(a) compares the accuracy values obtained for the above-mentioned datasets. For each dataset (on the x-axis), the results obtained by the same technique across different datasets are connected by a colored line; distinct colors correspond to different techniques. This representation clearly illustrates the relative behavior of all the evaluated methods.

Clearly, training ML models on the synthetic datasets results in a loss of accuracy, as expected. In fact, the results obtained for the "Original" dataset are slightly higher than those of the corresponding synthetic datasets. Nevertheless, the loss is not significant, approximately 2 % or less.

- Apart from the initial loss, the behavior of each single technique is very stable. Undeniably, even reducing the generation factor to 10 %, there is never a drop of accuracy. Also, surprisingly, for *SVM* and *NBayes* there is a slight improvement from 20 % to 10 %.
- The techniques that exhibit the poorest performance are *kNN* and *RTree*. Their accuracy is approximately 65 % on the original dataset, while it remains stable around 62 % on the synthetic ones. Thus, the loss in accuracy is about 3 %, which is not excessive.
- For the remaining techniques, the accuracy on the synthetic datasets is no lower than 68 %.

Specifically, note that *NBayes* exhibits only a limited loss of accuracy when moving from the Original dataset to the synthetic ones, remaining between 68 % and 70 %.

Apart from *SVM*, which performs slightly worse than the other techniques in this group, the other methods behave substantially in the same way on synthetic datasets, making them substantially indistinguishable.

The results show that the loss is always limited, even negligible in the case of *NBayes*. Consequently, it is actually possible to exploit small-sized synthetic datasets to train ML models in place of the large source dataset.

To further analyze the behavior, the F1-score was also evaluated. In fact, Fig. 3(b) illustrates these results.

The reader can notice that the loss is still contained for all the techniques. Nonetheless, the behavior of the F1-score is more variable than the accuracy. This is particularly evident for *NBayes*, which loses both precision and recall for a few small synthetic datasets.

5.2. Dataset DS2

The experiments were repeated for dataset $DS2$ in order to determine whether the synthetic datasets generated from a different source remain as close to the original distribution as those obtained from $DS1$. The results, in terms of classification accuracy, are shown in Fig. 4(a).

For $DS2$, a smaller number of synthetic datasets was considered. Specifically, apart from the original dataset, the regeneration factors of 150 %, 100 %, 50 %, and 10 % were considered,

As for $DS1$, on the original dataset the classification techniques show comparable accuracy.

Furthermore, the loss of accuracy that is observed is comparable as well. In our research work, in the worst cases, the difference is limited to 2 %. These results confirm that the described approach for generating synthetic datasets, which was implemented within *YABaGen* promises to be stable as far as its capability of reproducing the source data distribution is concerned.

As for the dataset $DS1$, the F1-score was evaluated for the dataset $DS2$. The achieved results are depicted in Fig. 4(b).

Similarly to dataset $DS1$, the F1-score exhibits a more variable behavior.

In particular, it is noteworthy that for *NBayes*, in a few cases the accuracy obtained on the synthetic datasets is even higher than that achieved on the original dataset. This effect can be explained by the fact that some outlier rows are not reproduced in the synthetic data, allowing the classification algorithm to learn the underlying probability distribution more precisely.

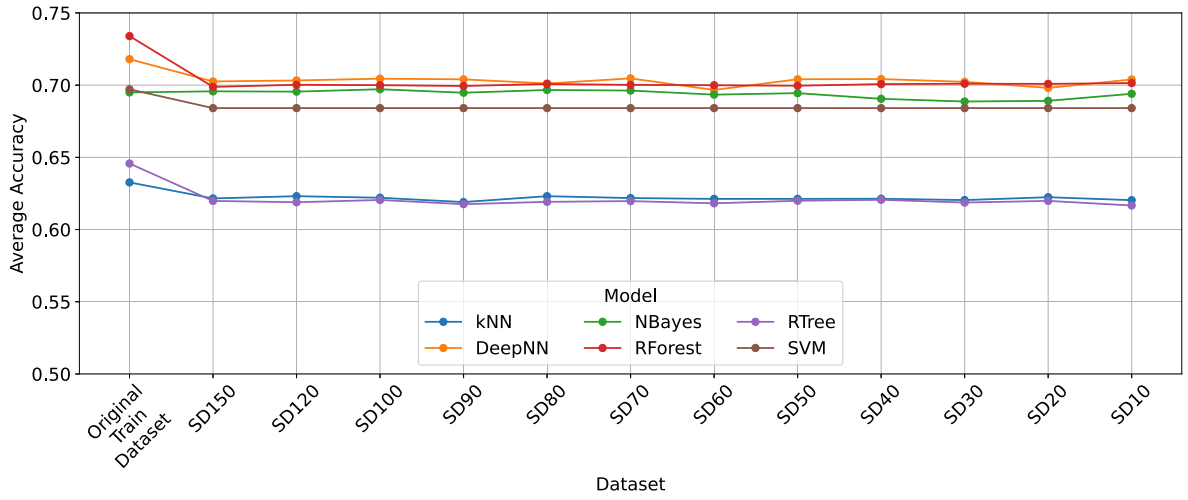
5.3. Comparing original and synthetic datasets

The results discussed so far, such as the limited loss in accuracy observed when ML methods are trained on synthetic datasets, are certainly positive. Nonetheless, to more thoroughly analyze the distance between the synthetic datasets and the original one, a "dataset-comparison method" was applied.

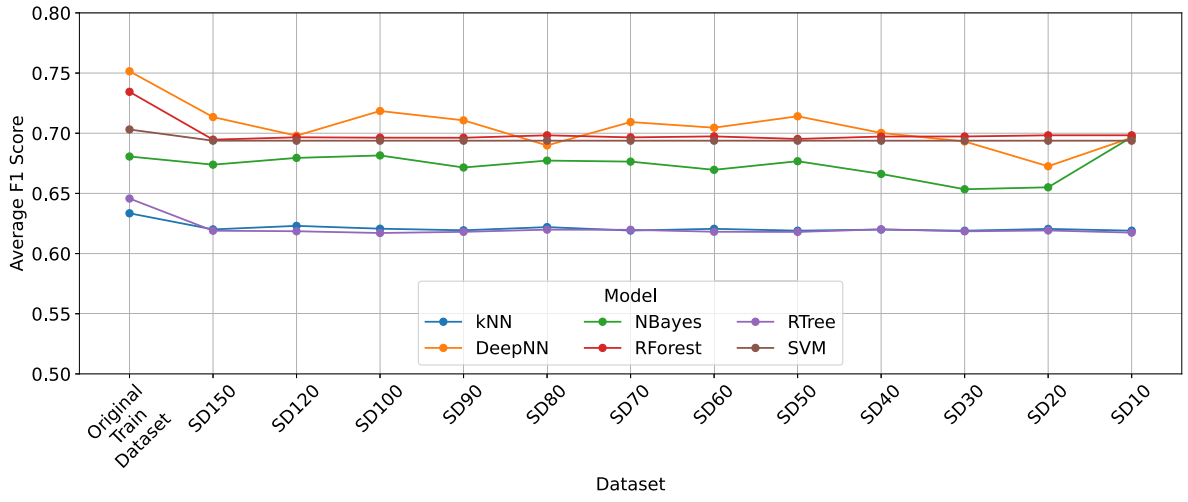
Specifically, this research exploited the method that was presented in [31]. The proposed technique compares two datasets on the basis of a statistical method, whose goal is to analyze and compare their distributions. The authors of Gretton et al. [31] base their method on a metric named "Maximum Mean Discrepancy" (MMD). The MMD score is in the range $[0, 1]$: if it is 0, the two compared datasets are identical; the greater the score, the larger the difference between them.

The strength of MMD lies in its independence from any specific family of probability functions. Therefore, it can be used to measure and compare the probability distributions of two generic datasets.

For both datasets $DS1$ and $DS2$, a set of comparisons was performed: the original dataset and the synthetic datasets $SD010$, $SD050$, $SD100$, and $SD150$ (generated by *YABaGen*) were cross-compared. Fig. 5 shows



(a) Dataset $DS1$ - Accuracy.



(b) Dataset $DS1$ - F1-score.

Fig. 3. Accuracy (top) and F1-score (bottom) for the dataset $DS1$. Lines connect results obtained for the same ML technique.

the heatmaps that were obtained. Specifically, Fig. 5(a) depicts the results for the dataset $DS1$, while Fig. 5(b) depicts the results for the dataset $DS2$.

As far as $DS1$ is concerned, the reader can see that the MMD values are very low, no greater than 0.170×10^{-3} when the original dataset is compared with the synthetic ones; moreover, comparing synthetic datasets with each other, the MMD value is no greater than 0.074×10^{-3}, which are extremely low values, meaning that the synthetic datasets are quite homogeneous.

Considering dataset $DS2$, the MMD values are much greater than those for $DS1$, where the worst MMD value is greater than 18×10^{-3}. Comparing the synthetic datasets with each other, the MMD values are generally very low, meaning that they are quite homogeneous, apart from $SD010$, which is very non-homogeneous; the explanation could be that $DS2$ is smaller than $DS1$, consequently $SD010$ is less representative of the original datasets than it is for $DS1$.

In order to perform a preliminary comparison with previous techniques for synthetic data generation, the tool *Synthetic Data Vault* which implements the technique named *Gaussian Copula* [32] was run on the dataset $DS1$. Fig. 6 shows the results of the comparisons. Explicitly, Fig. 6(a) reports the same heatmap depicted in Fig. 5(a) with a different color scale, i.e., the comparison performed on synthetic datasets generated by *YABaGen*. Then, Fig. 6(b) reports the heatmap for the tool *Synthetic Data Vault*. As the reader can observe, the two heatmaps adopt the same color scale.

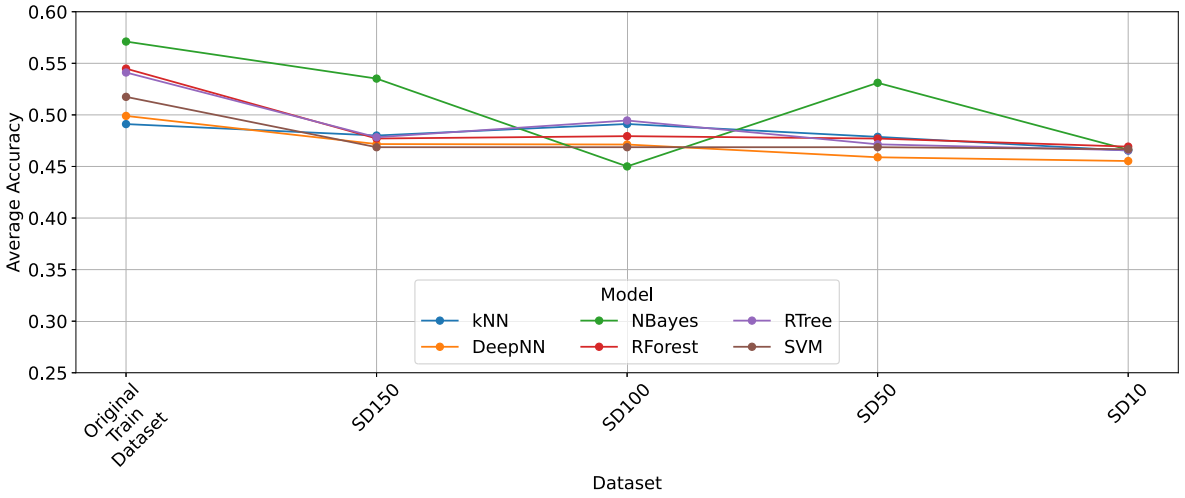
Moreover, please notice that the achieved MMD values are absolutely comparable. This means that *YABaGen* and *Synthetic Data Vault* are equally effective, as both are able to generate synthetic datasets that closely reproduce the original probability distribution. Consequently, it can be stated that *YABaGen* is as effective as the state-of-the-art in the field.

6. Evaluation: computational resources

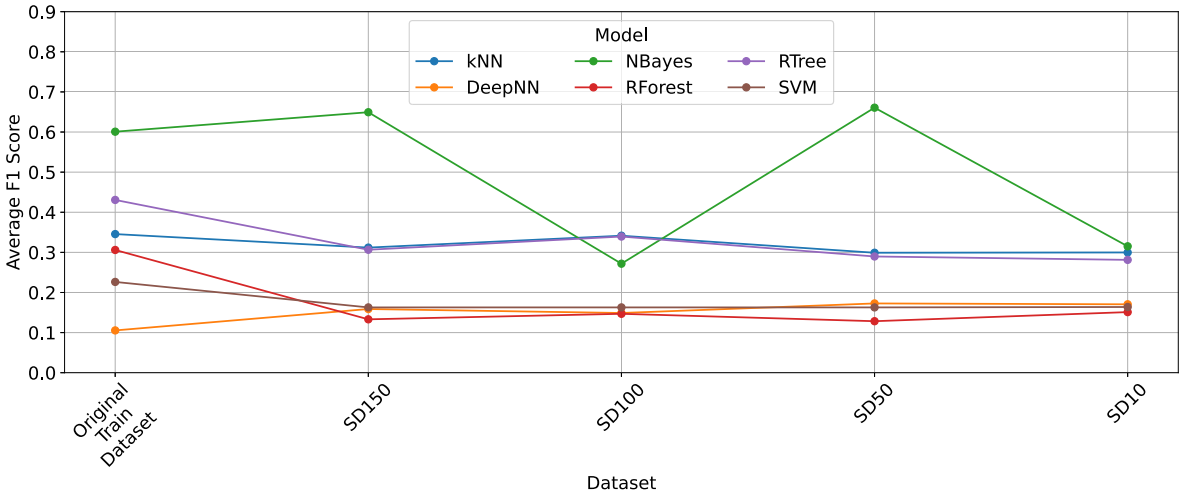
In order to have a complete understanding of the impact of *YABaGen*, the amount of computational resources needed by ML techniques is fundamental. In this way, considering both execution time and main memory usage can provide better insight into how the use of synthetic datasets affects ML techniques. The experiments were performed on the dataset $DS1$ and its derived synthetic datasets (see Section 4.2).

6.1. Execution times

Typically, the training phase of ML algorithms is perceived as the most expensive one, when the size of the training set is large.

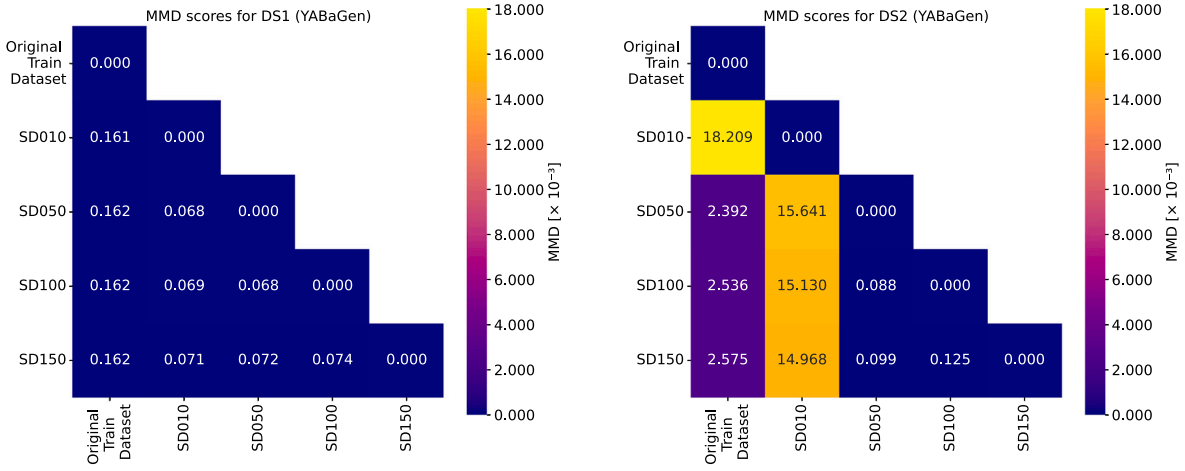


(a) Dataset *DS* 2 - Accuracy.



(b) Dataset *DS* 2 - F1-score.

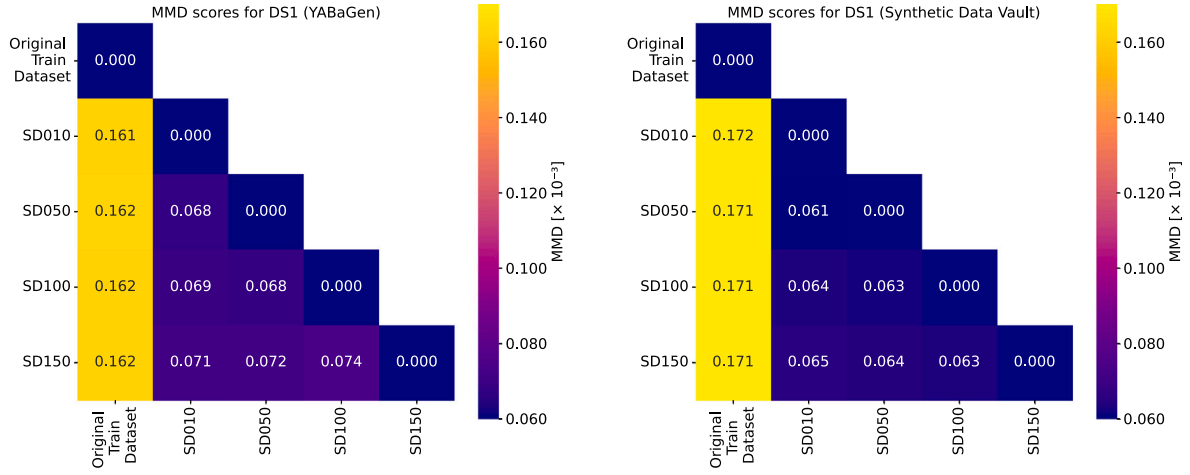
Fig. 4. Accuracy (top) and F1-score (bottom) for the dataset *DS*2. Lines connect results obtained for the same ML technique.



(a) MMD scores for the dataset *DS* 1.

(b) MMD scores for the dataset *DS* 2.

Fig. 5. MMD scores for synthetic datasets generated by *YABaGen*.



(a) MMD scores for dataset *DS*1 with *YABaGen*.

(b) MMD scores for dataset *DS*1 with *Synthetic Data Vault*.

Fig. 6. MMD scores - *YABaGen* vs *Synthetic data vault* for the dataset *DS*1.

Nevertheless, different techniques can exhibit different behavior, so the execution time in both phases is measured and studied. Each phase was replicated 3 times in order to calculate the average execution time.

Training Phase. The left-hand side of Fig. 7 reports the average execution times that were measured for training (on the dataset *DS*1) the ML models that were illustrated in Section 4 and reported in Table 1. The left-hand side of Fig. 7 depicts the results of the measurements: ML techniques were trained on the synthetic datasets generated from the dataset *DS*1 by *YABaGen*. The original dataset was not considered, since the dataset *SD100* has the same size as the original one. The left-hand side of Fig. 7 also reports the average execution time that was necessary to generate the synthetic datasets (gray line).

Fig. 7(a) depicts the lines for all the techniques. It is possible to notice that more or less all the curves are in the same area of the chart, except for the purple line, which corresponds to the *SVM* technique. This clearly shows how the algorithm at the basis of *SVM* is exponential. For the dataset *SD150* it needs around $67,000$ seconds, that is more than 18.5 hours, while the other techniques do not exceed 2000 seconds (i.e., a little more than half an hour). Consequently, to better show the behavior of the other techniques, Fig. 7(c) focuses on the range $[0, 2000]$ seconds.

From Fig. 7(c), the reader can notice how the time that was needed to train *DeepNN* (orange line) is slightly greater than the time needed to train *RForest* (red line). Interestingly, all the remaining curves are very close to the x-axis and are substantially indistinguishable. In summary, this means that the training time needed by the corresponding technique is significantly lower than the generation time.

In order to clearly see the remaining lines, Fig. 7(e) is further focused on the range $[0, 40]$ seconds. In particular, notice that *kNN* is substantially instantaneous, because it does not train any model.

Test Phase. Considering the execution time during the testing phase, the execution time for performing the phase on the test set of the dataset *DS*1 (see Section 4.2) is illustrated in the right-hand side of Fig. 7. In particular, Fig. 7(b) reports all the curves. Please, notice that the gray line does not appear, since reporting the time necessary to generate the synthetic dataset is unnecessary here.

While most of the techniques are located in the same area of the chart, at the bottom, there is one outlier. It is the curve for the *kNN* technique (blue line). Specifically, during the testing phase, *kNN* is linear, yet it is extremely expensive. This is due to the fact that *kNN* does not build an internal model during the training phase. Hence, during the test phase, the algorithm has to analyze the source dataset.

In order to better analyze the behavior of the remaining techniques, Fig. 7(d) focuses on the previous figure within the range $[0, 100]$ seconds.

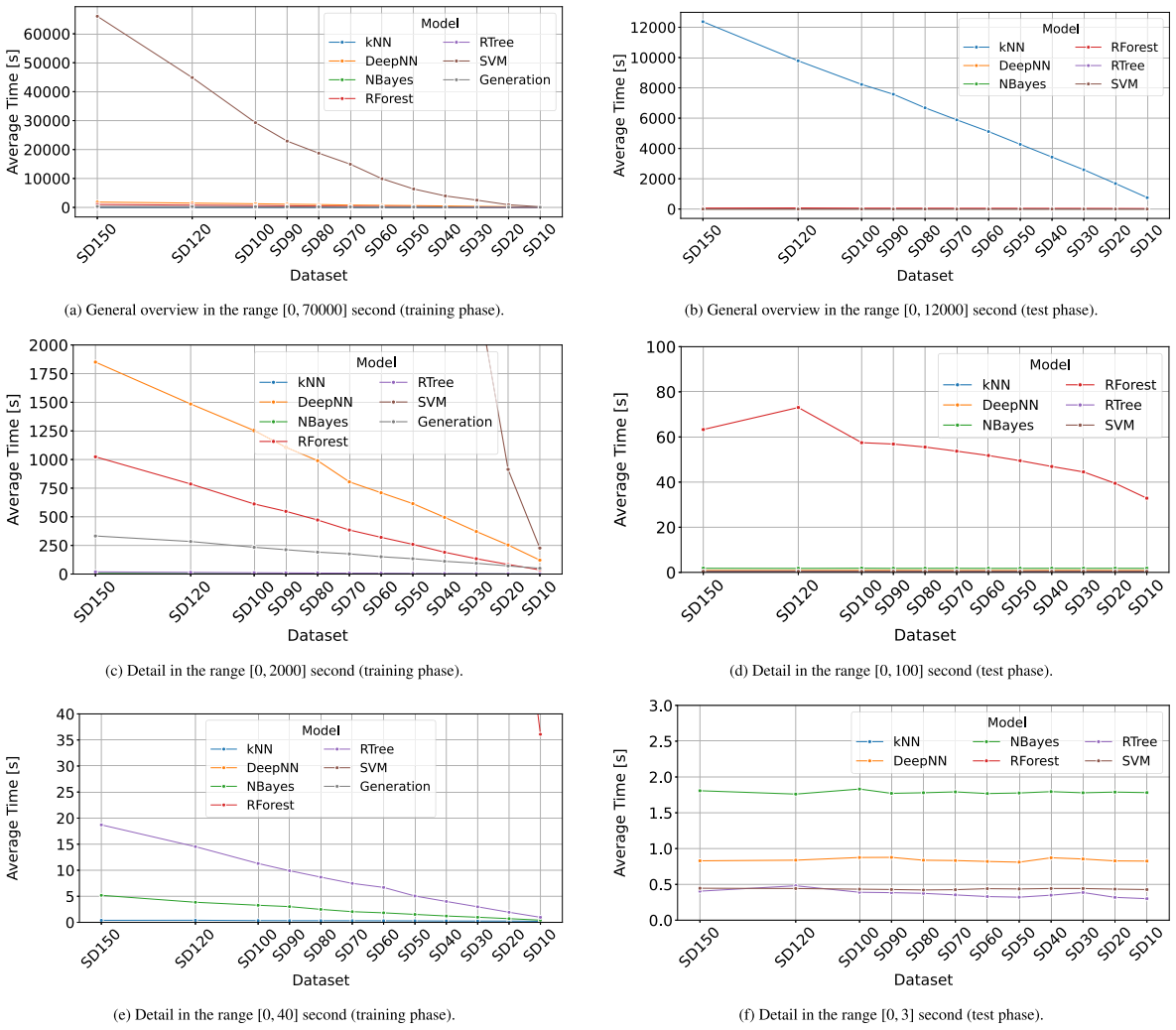
Note that only the red curve is higher than the other ones. Specifically, the red curve corresponds to *RForest*. The reason for its inefficiency, in relation to the other techniques whose curves are overlapped at the bottom is the large number of trees that compose the "Forest"; this structure of the model tends to make this technique inefficient during the test phase. Nonetheless, it is much more efficient than *kNN*.

In order to complete the analysis, Fig. 7(f) further focuses on the range $[0, 2.5]$ seconds. The reader can notice that the fastest techniques during the test phase are *RTree* and *SVM*, which are substantially comparable and require the same execution time, i.e., around 0.5 second. Remember that, during the test phase, the same test set was used; thus, the difference lies in how the model was trained (denoted by the x-axis). Overall, it can be argued that the size of the model does not vary substantially when the size of the training set changes. As a result, the measured execution times during the test phase, performed on the same test set, remain stable. In contrast, the size of the model changes for *RForest* (see Fig. 7(d)): the curve shows a descending trend, meaning that the size of the model is larger for larger training sets. Analogously (see Fig. 7(b)), *kNN* shows a purely linear trend, because the model is the training set itself.

Overall View. In order to understand the impact of the training and test phases, in relation to the time needed to generate synthetic data, an orthogonal analysis is performed hereafter. Specifically, Fig. 8 reports the cumulative execution times for each single technique. Each bar is obtained by stacking (in the order) : (i) the execution time for generating the synthetic dataset (in gray), (ii) the execution time to train the model (using the same color previously adopted for the specific technique in Fig. 7, (iii) the execution time during testing (in black). The first bar on the left corresponds to the execution times measured on the original dataset (thus, the gray component, i.e., generating the synthetic dataset, is not present).

- Fig. 8(a) shows the execution times that were measured for the *RForest* technique. It is possible to see that the contribution of the test phase (black) is often marginal in comparison with the training phase, at least for datasets larger than *SD020*.

  Evaluating the cumulative contributions, it appears that the generation of the synthetic datasets becomes effective when the size of synthetic datasets is less than 60 % the original size. Indeed, after this point, the training times significantly reduce, compensating the generation time. Consequently, the limited loss of accuracy is a reasonable price to pay, this way gaining a much faster training phase, even cumulated with the generation time.



(a) General overview in the range [0, 70000] second (training phase).

(b) General overview in the range [0, 12000] second (test phase).

(c) Detail in the range [0, 2000] second (training phase).

(d) Detail in the range [0, 100] second (test phase).

(e) Detail in the range [0, 40] second (training phase).

(f) Detail in the range [0, 3] second (test phase).

Fig. 7. Execution times during the training (left) and test (right) phase for the dataset $DS1$.

Consequently, working on small synthetic datasets is very beneficial for classifiers that are based on *RForest*.

- Fig. 8(c) depicts the cumulated execution times for *RTree*.

The reader can notice how the generation times (gray bars) are significantly greater than training and test times. In particular, notice that the test time is so small compared with the other times that there is no trace of black bars (that depict the test time).

Thus, the effectiveness of working on the synthetic datasets is to store a small-size dataset for later processing, not for saving time.

- Fig. 8(e) depicts the cumulative times for *SVM*.

In this case, with respect to *RTree*, the opposite situation occurs, i.e., the training times strongly dominate the chart. Indeed, almost no trace of the gray (generation time) and black (test time) bars can be seen, meaning that they are irrelevant in terms of relative importance.

Consequently, it is possible to say that, in case of *SVM*, the greater the reduction of the dataset, the greater the impact on the training time. Therefore, the best results in terms of reduction of execution time are obtained with the smallest dataset *SD010*. The practical effect is that *SVM* becomes practically exploitable, yet paying a small loss of accuracy.

- Fig. 8(b) depicts the cumulative execution times for *kNN*.

The reader can see that the execution times are dominated by the testing times (black bars). The other execution times, such as generation times, in gray, and training time, in blue, are irrelevant. This is due to the fact that a *kNN* classifier does not build a model, but uses the entire training set during the test phase.

Consequently, the greatest effects (in terms of execution times) of generating a synthetic dataset are obtained with the smallest synthetic dataset *SD010*. The test phase becomes practically exploitable, with the price of a small loss for accuracy. Remember that the training phase is negligible, because no model is built during the training phase.

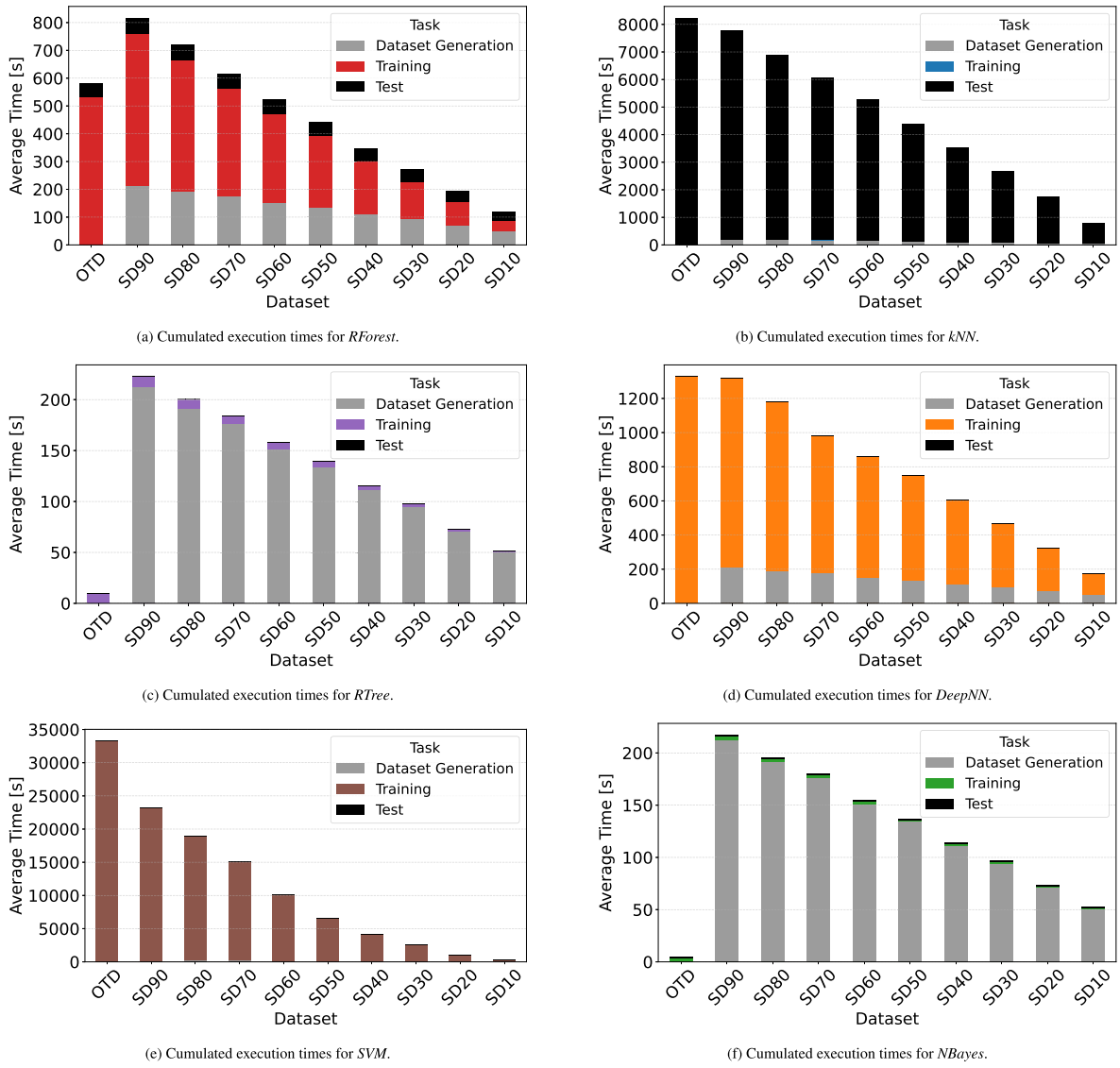
- Fig. 8(d) cumulatively depicts the execution times for *DeepNN*.

The first thing that the reader can notice is the absence of the black bars: this means that the test time is negligible in comparison with training time and generation time. Consequently, introducing the synthetic dataset should affect only the training time.

Indeed, this is confirmed by the chart: the dominant contribution is the training time, but the introduction of the synthetic datasets has positive effect even with *SD080*. Reducing the size of the training set causes a significant reduction of the training time. This reduction compensates the time needed for generating the synthetic dataset.

Thus, with small synthetic datasets, the cumulated generation and training times become significantly smaller, making them feasible for fast computation.

- Fig. 8(f) depicts the cumulated execution times for *NBayes*.



(a) Cumulated execution times for *RForest*.

(b) Cumulated execution times for *kNN*.

(c) Cumulated execution times for *RTree*.

(d) Cumulated execution times for *DeepNN*.

(e) Cumulated execution times for *SVM*.

(f) Cumulated execution times for *NBayes*.

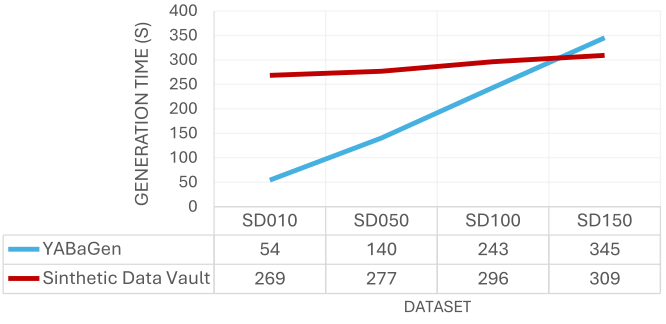
Fig. 8. Cumulated execution times.

The main characteristics of this technique are very quick construction of the model during the training phase and quick test phase. In this case, the chart is dominated by the generation time (gray bar) of the synthetic datasets. Thus, adopting synthetic datasets can be useful for storing a compact version of the training datasets for future exploitation, not for improving execution times of the classifier.

Comparison with Other SDG. *YABaGen* was also compared with *Synthetic Data Vault* as far as execution times for generating synthetic datasets are concerned: when *Synthetic Data Vault* was run to generate (from the dataset $DS1$) the synthetic datasets that were compared in Section 5.3, execution times were measured as well.

Fig. 9 reports the comparison results. Moreover, the table at the bottom shows the execution times (in seconds) required to generate the synthetic datasets $SD010$, $SD050$, $SD100$, and $SD150$. The upper row reports the times required by *YABaGen*, whereas the lower row reports those required by *Synthetic Data Vault*. The chart in the upper part of the figure plots these values, where the blue line corresponds to *YABaGen* and the red line corresponds to *Synthetic Data Vault*.

The reader can see that *YABaGen* is significantly faster than *Synthetic Data Vault* for generating small synthetic datasets. It appears that *Synthetic Data Vault* is less efficient than *YABaGen* when it trains the model on the basis of which it generates synthetic datasets. In this respect, *YABaGen* is very efficient when it trains the Bayesian Network from which it generates the synthetic datasets. In contrast, once the model, *YABaGen* shows that it is less efficient than *Synthetic Data Vault*; indeed, when very large synthetic datasets are generated, *Synthetic Data Vault* is more efficient. Definitely, in the scenario in which small



	SD010	SD050	SD100	SD150
YABaGen	54	140	243	345
Sinthetic Data Vault	269	277	296	309

Fig. 9. Generation time: *YABaGen* vs *synthetic data vault* (in seconds).

(a) Occupied memory during training phase.

(b) Occupied Memory during test phase.

(c) Occupied memory during training phase (focused).

(d) Occupied memory during test phase (focused).

Fig. 10. Occupied memory during the training (left) and test (right) phases.

synthetic datasets are generated, *YABaGen* is significantly faster than *Synthetic Data Vault*.

6.2. Occupied memory

A second feature to consider in understanding the impact of generating and working on synthetic datasets is the memory occupied during the various steps of the process based on *YABaGen*, working on the dataset *DS*1. In the remainder of this section, the occupied memory during the training phase and the test phase is analyzed.

Training Phase. The left-hand side of Fig. 10 reports the curves that depict the total amount of main memory that was occupied during the training phase of ML models. In addition, the gray curve depicts the memory occupied by the *YABaGen* tool during the generation of the corresponding synthetic dataset.

Fig. 10(a) depicts all the curves. The reader can see that *RForest* (red line) is the technique that requires significantly more memory than the other techniques. In fact, it occupied more than 6 GByte, while the other techniques did not exceed 1.5 GByte. Notice that *SVM* (purple line) was the most inefficient in terms of execution times, but it does not occupy so much main memory, meaning that its inefficiency is due to its algorithm.

The reader can also see that the *YABaGen* tool (gray curve) occupied a large volume of main memory as well. For sure, one reason is that both the input and the output datasets are kept in main memory. In particular, that is necessary for the output datasets, because they are saved in ARFF format. So, for doing that, the Java library prepares the output dataset in main memory and later saves it into a file. Indeed, the reader can see that occupied memory drops from 4.6 GByte (for *SD150*) to 1 GByte (for *SD010*), meaning that it is the preparation of the output synthetic dataset that occupies a large amount of main memory. The spike in the gray curve is due to the unclear behavior of the Java Virtual Machine.

In order to better visualize and study the behavior of the other techniques, Fig. 10(c) is focused on the range $[0, 1.6]$ GByte. The reader can notice how the curves, more or less, follow the same trend, obviously reducing the necessary amount of main memory as the size of the training set decreases. Apart from some outliers, due to the internal mechanisms of the Java Virtual Machine that executes *YABaGen*, the trend is consistent.

Notice that even *kNN*, which is extremely fast during the training phase, still occupies a significant amount of memory, which is comparable with other techniques. This means that it actually loads the training set into main memory, to perform some quick analysis on it.

The charts on the left-hand side of Fig. 10 clearly show that reducing the size of the training set significantly reduces the amount of main memory necessary to perform the training phase. Consequently, working on small synthetic datasets instead of the large original one could be beneficial when the available main memory is not large (for example, on mobile devices).

Test Phase. The right-hand side of Fig. 10 reports the curves that depict the amount of main memory that was utilized during the test phase.

Then, Fig. 10(b) reports all the curves. The reader can notice that *RForest* (red line) still uses much more main memory than the other techniques: more than 5 GByte. This means that the model (i.e., a pool of classification trees) is really large and expensive to represent. Consequently, *RForest* is not suitable when the main memory is limited.

In order to complete the analysis for the remaining techniques, Fig. 10(d) is focused on the range $[0, 1.6]$ GByte. In this case, the curves are generally flat, meaning that the size of the model is not affected by the size of the training set.

The only exception is *kNN* (blue line), for which the model is the training set itself. This consideration explains why the curve shows a decreasing trend.

Considering *NBayes* (green curve), it is possible to see that the model occupies more than 1 GByte. On the opposite side, the model of *DeepNN* (orange curve) occupies only 0.4 GByte, meaning that the model generated by this neural network is quite compact. Consequently, it can be exploited for performing the test phase on mobile devices. Similarly, *SVM* (purple curve) confirms that its model is quite compact: once trained (it needs a very large training time), it can be effectively exploited on mobile devices as well.

7. Discussion

In order to properly conclude the evaluation, it is worth discussing the main outcomes of the analysis that were presented in Sections 5 and 6. In particular, the discussion will provide direct answers to the research questions that were introduced in Section 1.

Q1: Loss of Accuracy. The limited loss of accuracy that was observed while training ML models on a synthetic training set in place of the original dataset is a very good result. This means that the *Bayesian Generation*

technique is actually effective in maintaining the probability distribution in the synthetic datasets very close to the original one, even for small synthetic datasets. The comparison with a state-of-the-art DSG tool (*Synthetic Data Vault*) also showed that *YABaGen* is actually as good as previous SDG tools.

Although larger datasets often improve learning accuracy, the experiments show that smaller synthetic datasets can preserve comparable model performance. This behavior arises from the probabilistic nature of the Bayesian generation process of *YABaGen*, which maintains the joint data distribution of the original dataset while inherently filtering out noise and outliers during synthesis. As a result, the generated datasets capture the essential statistical structure of the data with reduced variability. Nevertheless, when the regeneration factor becomes too aggressive (i.e., too small), part of the information contained in the original dataset may be lost, leading to a gradual degradation in predictive performance. Therefore, while maintaining very large synthetic datasets is often unnecessary, the choice of the regeneration factor must balance computational efficiency with representational fidelity.

Q2: Impact of small synthetic datasets on computational resources for training/testing. Discussing this point requires a more articulate argument.

- Some ML techniques are intrinsically inefficient during training (e.g., *SVM*), Clearly, training the model on a small synthetic dataset certainly makes it possible to run the training phase of this technique quicker than on the original dataset, thus, saving time, with a limited loss of accuracy.
- Apart from *SVM*, the other ML models are trained in a shorter time than the one necessary to generate the synthetic training dataset.

  Nonetheless, the stability of the accuracy of all the models, once trained on the synthetic datasets (independently of the size) indicates that, for most of the techniques, the time required to generate even the smallest synthetic datasets is more than compensated by the reduction of training time. Only *RTree* and *NBayes* do not receive any benefit, because they are extremely fast.
- Considering the test phase, it is clear that *kNN* benefits the most from operating on the small synthetic dataset. However, it remains significantly slower than the other techniques and requires the largest amount of main memory, i.e., approximately 2 GByte.

  RForest also gets benefits during the test phase in terms of execution times, while the other techniques remain unaffected (during the test phase).
- As far as the occupied memory is concerned, on one hand, during the training phase, it decreases significantly for all techniques when training is performed on the small synthetic datasets.

  On the other hand, during the test phase, apart from *RForest* and *kNN*, the occupied memory is stable. This means that the size of the models is not significantly affected by the size of the training set.

Q3: Benefits provided by the effort of generating small synthetic datasets. Generating a small synthetic dataset can be beneficial from many points of view.

1. It is possible to store a small dataset in place of the large source dataset.
2. It is possible to train ML models, saving time for analysis tasks.
3. For some techniques, such as *DeepNN*, *SVM*, and *RTree*, the trained model occupies substantially the same amount of memory, i.e., 0.4/0.6 GByte. Consequently, with these models, the testing phase can be easily performed either on mobile devices or onboard industrial machines.

   The most notable effect is observed for *kNN*. Since the model is the training set itself, *kNN* also becomes suitable for mobile or industrial devices.

Although the primary focus of *YABaGen* is on generating compact synthetic datasets for efficient ML training, the method could potentially be extended to data augmentation tasks. If the regeneration factors used during the generation process are chosen appropriately, the resulting synthetic datasets can represent additional observations that follow the underlying probabilistic distribution of the original data, effectively providing "new" samples while preserving statistical properties.

8. Conclusions and future work

To wrap up the findings and contributions of this study, this section provides the key conclusions and discusses possible avenues for further exploration.

8.1. Conclusions

The paper stems from the development of a technique named *Bayesian Generation* for generating synthetic datasets. The technique, originally presented in [5], is implemented within the *YABaGen* software tool, which is written in Java and designed for potential inclusion in the WEKA suite.

Building on this premise, the contribution of this paper lies in analyzing the effects of training and practically applying ML techniques using small synthetic datasets instead of the original large dataset. Firstly, the study compares the effectiveness of ML models trained on small synthetic datasets with that of the same models trained on the large original dataset. Secondly, execution times and memory usage were collected for synthetic dataset generation, as well as for the training and testing phases performed with the considered techniques. Finally, the study examines these data from multiple perspectives using a set of plots.

The behavior of *YABaGen* was also compared with a state-of-the-art tool for synthetic data generation, namely *Synthetic Data Vault*. Regarding the similarity between the synthetic datasets produced by the two tools, the analysis based on the MMD metric indicates that both achieve a comparable degree of similarity to the corresponding original datasets. The accuracy measured by classifiers trained on synthetic datasets generated by the two tools exhibits similar behavior, particularly for the larger source dataset $DS1$. In terms of execution times, the *YABaGen* tool is significantly faster than *Synthetic Data Vault* when small datasets are generated. These results suggest that *YABaGen* can be effectively and practically adopted as an alternative to existing SDG tools.

These results are noteworthy, as they provide a perspective that has been relatively unexplored in this field, while also addressing the 3 research questions presented in Section 1: (Q1) the loss of accuracy is limited, and the behavior of ML models on synthetic datasets is stable, even for small synthetic datasets; (Q2) most ML techniques benefit when training is performed on small synthetic datasets, both in terms of execution time and occupied memory, often compensating for the effort to generate the synthetic dataset; (Q3) the expected practical impact could be significant; in fact, small synthetic datasets could be stored in place of the original large datasets, and several techniques would become feasible for executing the testing phase on mobile or industrial devices.

Nonetheless, the study of execution time and memory usage can offer researchers and practitioners valuable insights into the practical applicability of each single technique in contexts with limited computational resources, such as mobile and industrial devices, both when used with and without synthetic data generation. This issue could be further explored by adopting specific evaluation metrics, such as in [33].

8.2. Future work

Regarding the experimental results achieved, together with the discussions and conclusions presented in this study, it is evident that several avenues for future research emerge. These potential directions aim to

deepen the understanding of the observed phenomena and to extend the applicability of the proposed approaches in broader or more complex scenarios, such as those listed below.

The first line of future work that the authors plan to pursue is to apply *Bayesian Generation* and *YABaGen* in an industrial environment, with datasets that come from industrial devices. There is a typical need in such environments to ensure the quality of products, while detecting any potential issues that could occur during processes, for example, in a foundry.

The second research direction concerns the development of an enhanced version of *Bayesian Generation* capable of incrementally incorporating updates to the source dataset. The idea is to allow the system to progressively enrich either the previously generated synthetic dataset or the previously trained model. In this way, an incremental approach would enable *Bayesian Generation* to adapt to evolving data without requiring full regeneration or retraining from scratch.

The third direction focuses on privacy-preserving applications, exploring the feasibility of applying *Bayesian Generation* under strict privacy constraints. This aspect is particularly relevant in regions with restrictive privacy regulations such as Europe.

Finally, an important avenue for future work involves a more comprehensive benchmarking of *YABaGen* against other established synthetic data generators, including *SDV* (*Synthetic Data Vault*) and *CTGAN* models. Specifically, comparisons in terms of generation speed, scalability, and data fidelity would provide valuable insights into the practical positioning of *YABaGen* within the broader SDG landscape. This accurate benchmarking is a necessary step toward further validating the practical applicability of *YABaGen* across diverse datasets and ML scenarios. This task is planned as a primary direction for future work, during which the impact of using synthetic datasets for training complex classification models, such as CNNs and Transformers, will be studied.

CRediT authorship contribution statement

Paolo Fosci: Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Conceptualization. **Javier Nieves:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Conceptualization. **Giuseppe Psaila:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Formal analysis, Conceptualization. **Jacopo Boffelli:** Writing – original draft, Visualization, Software. **Pablo Garcia Bringas**: Supervision, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships that may be considered potential competing interests:

1. For Paolo Fosci (University of Bergamo) and Giuseppe Psaila (University of Bergamo) This study was funded by the European Union - NextGenerationEU, in the framework of the GRINS - Growing Resilient, Inclusive and Sustainable project (GRINS PE00000018 – CUP F83C22001720001). The views and opinions expressed are solely those of the authors and do not necessarily reflect those of the European Union, nor can the European Union be held responsible for them.

2. For Javier Nieves currently working for Atzerlan.

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

Acknowledgments

This study was funded by the European Union - NextGenerationEU, within the framework of the GRINS - Growing Resilient, INclusive and Sustainable project (GRINS PE00000018 – CUP F83C22001720001). The views and opinions expressed are solely those of the authors and do not necessarily reflect those of the European Union, nor can the European Union be held responsible for them.

Data availability

Data will be made available on request.

References

[1] J. Wu, Y. Yang, X.U.N. Cheng, H. Zuo, Z. Cheng, The development of digital twin technology review, in: 2020 Chinese Automation Congress (CAC), IEEE, 2020, pp. 4901–4906.
[2] Z. Lv, Digital twins in industry 5.0, Research 6 (2023) 0071.
[3] M. Bertolini, D. Mezzogori, M. Neroni, F. Zammori, Machine learning for industrial applications: a comprehensive literature review, Expert Syst. Appl. 175 (2021) 114820.
[4] M.F. Uddin, N. Gupta, et al., Seven V's of big data understanding big data to extract value, in: Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education, IEEE, 2014, pp. 1–5.
[5] P. Fosci, J. Nieves, G. Psaila, P.G. Bringas, Bayesian generation of synthetic data, in: International Conference on Soft Computing Models in Industrial and Environmental Applications, Springer Nature Switzerland Cham, 2024, pp. 181–193.
[6] P. Ojeda, M.E. Garcia, A. Londoño, N.-Y. Chen, Monte Carlo simulations of proteins in cages: influence of confinement on the stability of intermediate states, Biophys. J. 96 (3) (2009) 1076–1082.
[7] H.T. MacGillivray, R.J. Dodd, B.V. McNally, J.F. Lightfoot, H.G. Corwin, S.R. Heathcote, Monte-Carlo simulations of galaxy systems: I: the local supercluster, Astrophys. Space Sci. 81 (1982) 231–250.
[8] M. Endres, A. Mannarapotta Venugopal, T.S. Tran, Synthetic data generation: a comparative study, in: Proceedings of the 26th International Database Engineered Applications Symposium, 2022, pp. 94–102.
[9] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, SMOTE: synthetic minority over-sampling technique, J. Artif. Intell. Res. 16 (2002) 321–357.
[10] B. Nowok, synthpop: an R package for generating synthetic versions of sensitive microdata for statistical disclosure control, Tech. Rep., Administrative Data Research Centre …, 2015.
[11] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, Adv. Neural Inf. Process. Syst. 27 (2014).
[12] D.P. Kingma, M. Welling, et al., Auto-encoding variational Bayes, 2013.
[13] L. Xu, M. Skoularidou, A. Cuesta-Infante, K. Veeramachaneni, Modeling tabular data using conditional GAN, Adv. Neural Inf. Process. Syst. 32 (2019).
[14] A. Borji, Pros and cons of GAN evaluation measures, Comput. Vis. Image Underst. 179 (2019) 41–65.
[15] J. Ho, A. Jain, P. Abbeel, Denoising diffusion probabilistic models, Adv. Neural Inf. Process. Syst. 33 (2020) 6840–6851.
[16] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, A.M. Lopez, The synthia dataset: a large collection of synthetic images for semantic segmentation of urban scenes, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 3234–3243.
[17] L. Yu, W. Zhang, J. Wang, Y. Yu, SeqGAN: sequence generative adversarial nets with policy gradient, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31, 2017, pp. 2852–2858.
[18] S. Floyd, M. Warmuth, Sample compression, learnability, and the Vapnik-Chervonenkis dimension, Mach. Learn. 21 (3) (1995) 269–304.
[19] N. Littlestone, M.K. Warmuth, Relating data compression and learnability, Tech. Rep., Citeseer, 1986.
[20] A. Blumer, A. Ehrenfeucht, D. Haussler, M.K. Warmuth, Learnability and the Vapnik-Chervonenkis dimension, J. ACM 36 (4) (1989) 929–965.
[21] I.H. Witten, E. Frank, Data mining: practical machine learning tools and techniques with Java implementations, ACM SIGMOD Rec. 31 (1) (2002) 76–77.
[22] S.J. Russell, Norvig, Artificial Intelligence: A Modern Approach, Second ed, Prentice Hall, 2003.
[23] N. Friedman, D. Geiger, M. Goldszmidt, Bayesian network classifiers, Mach. Learn. 29 (2) (1997) 131–163.
[24] R. Rubinstein y, D.P. Kroese, Simulation and the Monte Carlo Method, John Wiley & Sons, 2016.
[25] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (1995) 273–297.
[26] T.M. Cover, P.E. Hart, Nearest neighbor pattern classification, IEEE Trans. Inf. Theory 13 (1), 21–27, 1967.
[27] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32.
[28] Y. Amit, D. Geman, Shape quantization and recognition with randomized trees, Neural Comput. 9 (7) (1997) 1545–1588.
[29] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–536.
[30] D.D. Lewis, Naive (Bayes) at forty: the independence assumption in information retrieval, in: European Conference on Machine Learning, Springer, 1998, pp. 4–15.
[31] A. Gretton, K.M. Borgwardt, M.J. Rasch, B. Schölkopf, A. Smola, A kernel two-sample test, J. Mach. Learn. Res. 13 (1) (2012) 723–773.
[32] N. Patki, R. Wedge, K. Veeramachaneni, The synthetic data vault, in: 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), IEEE, 2016, pp. 399–410.
[33] M. Ali, A. Baqir, G. Psaila, S. Malik, Towards the discovery of influencers to follow in micro-blogs (twitter) by detecting topics in posted messages (tweets), Appl. Sci. 10 (16) (2020) 5715.