Received March 13, 2020, accepted April 22, 2020, date of publication May 6, 2020, date of current version May 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2991949

Synthetic Datasets Generator for Testing Information Visualization and Machine Learning Techniques and Tools

SANDRO DE PAULA MENDONÇA, YVAN PEREIRA DOS SANTOS BRITO, CARLOS GUSTAVO RESQUE DOS SANTOS, RODRIGO DO AMOR DIVINO LIMA, TIAGO DAVI OLIVEIRA DE ARAÚJO, AND BIANCHI SERIQUE MEIGUINS
Post-Graduate Program of Computer Science, Universidade Federal do Pará, Belém 66075-110, Brazil

Corresponding author: Carlos Gustavo Resque dos Santos (carlosresque@ufpa.br)

This work was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior–Brazil (CAPES)–Finance Code 001–and in part by the Pró-Reitoria de Pesquisa e Pós-Graduação (PROPESP/UFPA).

ABSTRACT Data generators are applications that produce synthetic datasets, which are useful for testing data analytics applications, such as machine learning algorithms and information visualization techniques. Each data generator application has a different approach to generate data. Consequently, each one has functionality gaps that make it unsuitable for some tasks (e.g., lack of ways to create outliers and non-random noise). This paper presents a data generator application that aims to fill relevant gaps scattered across other applications, providing a flexible tool to assist researchers in exhaustively testing their techniques in more diverse ways. The proposed system allows users to define and compose known statistical distributions to produce the desired outcome, visualizing the behavior of the data in real-time to analyze if it has the characteristics needed for efficient testing. This paper presents in detail the tool functionalities and how to create datasets, as well as a usage scenario to illustrate the process of data creation.

INDEX TERMS Synthetic dataset generator, benchmark datasets creation, data creation system.

I. INTRODUCTION

The ideal scenario for testing machine learning algorithms and information visualization techniques is to use real data. However, obtaining the data can be a relevant problem, since data may require a prolonged time to get, have associated costs, and have privacy concerns. In this context, the researchers are compelled to reuse the same, old, or well-known dataset to perform the tests. As an effort to mitigate these issues, researchers are either manually creating synthetic datasets or proposing applications that support this task. The researchers use these applications to have better control of the data characteristics, so they can create datasets to attack specific problems, such as outlier detection, missing values, and noisy information. [1], [2].

Synthetic data applications are commonly referred to as data generators, and they work by manipulating descriptive information of the data through mathematical formulas, probability distribution functions, category sets, and other generators. In some generators, users can easily share a blueprint of the generated dataset by saving a description of generators, which are usually lightweight files, instead of concrete data points.

One of the benefits of having a synthetic dataset generator is controlling data characteristics such as patterns, trends, data type, data format, outliers, dimensions, or missing values [3]. Data generators control data aspects so that their characteristics fit a specific problem, providing a diversity of slightly different datasets to exhaustively test visualization techniques or machine learning algorithms in a controlled manner [4], [5]. For example, in some generators, researchers can analyze not only if the technique is robust to the presence of outliers, but also what threshold of outlier proportion the technique can effectively handle.

However, no perfect data generator application exists. Every tool has its limitations, and as more of them are being created, the harder it becomes to choose one that fits the problem at hand. This difficult happens because the limitations are

The associate editor coordinating the review of this manuscript and approving it for publication was Shahzad Mumtaz.

scattered across many applications, so each one of existent data generators—while able to fills some gaps—leaves others untreated.

Hence, this work presents an application that generates synthetic datasets, providing a hub of functionalities that makes the application flexible enough to fills several gaps in previous approaches. The proposed data generator uses a combination of known distributions—called generators—that includes random distributions, data sequences, correlation functions, and modifiers. This approach enables the creation of data that can be applied in many domains, gradual changes in data characteristics by modifying parameters of generators, and creation of complex data distributions by combining a sequence of known ones.

Additionally, the proposed data generator supports visual feedback-driven design by plotting a sample of the current data model, and a faster way to collaborate and present data by sharing a lightweight description of the data instead of a large dataset file. The application is an open-source project and is available on the authors' research laboratory website.[1]

This paper is an extended version of a previous work [6], adding in-depth details about the functionalities and implementations present in the proposed data generator. New generators have been added since then, as well as new built-in visualization techniques. Additionally, a new usage case scenario is also presented to illustrate how researchers can test algorithms and techniques using synthetic datasets in the context of machine learning.

The next sections of this paper introduce the related works (section II), present the proposed system (section III), demonstrate usage scenario for the application (section IV), and conclude the work, providing future research directions (section V).

II. RELATED WORKS

Generation of synthetic datasets for testing has importance in many areas of computing, such as data visualization, data mining, software engineering, and artificial intelligence. Sran Popić *et al.* [7] wrote a survey about works in the area of synthetic data generation that focuses on application testing, highlighting the system architectures and the intended usage of the applications, showing the pros and cons of the surveyed techniques. Demillo and Offut [8] have described a failure-based application to generate synthetic data units for performing tests for software modules. Even in the area of evolutionary computing, there are works of genetic algorithms which generate data for software tests [9], [10].

Albuquerque *et al.* [11] have described a framework to generate multi-dimensional data. The user can build a representation of the desired data by manipulating statistical distributions through the graphical interface. However, [11] is limited to integer and floating numbers, not addressing the generation of categorical data. Moreover, [11] did not mention any way to preview the data, during the statistical distribution setup, to show the possible behavior of the generated data.

Wang *et al.* [12] have presented an application where it is possible for the user to scribble the distribution for the data manually. Thus, the system creates the data model of the generators based on what the user has drawn. Kwon *et al.* [13] used a similar approach, making use of design-based interactions to guide the creation of the data visualization with many dimensions according to the user's level of knowledge.

Liu [14] have created a synthetic data generator for assessing learning rules classification. The work generates learning rules based on the attributes entered by the user to build relationships between these attributes, and the technique used for the data was decision tree algorithms. Other similar works have appeared in the literature proposing data synthesizers for testing in data mining tools [15], [16] [17]. These works generate data for testing in data mining tools since obtaining real data can be very costly or limited by privacy rights. However, there are jobs that produce data for specific problems such as [18] who wrote a paper on the data generation system for health care applications, limiting the generation of new data only for such cases.

García and Millán [19] have created a system to generate synthetic data that can be used by a wide range of scientific areas. The authors compared their application with the tools that already exist in the market and have shown the pros and cons of their generator system. The software is available in a free version.

In some cases, applications generate synthetic data related to network data [20]. For instance, Brodkorb *et al.* [21] proposed the generation of synthetic network data with geo-location connected to the nodes. In this way, the user can explore the generated network through interaction with the map displayed and adjust the results obtained later.

Kofinas *et al.* [22] have created a methodology to generate synthetic data for simulating water consumption in two cities. The implemented approach registers the randomness of the daily water consumption by a local residence and validates the data generated by this methodology through validation algorithms which compare several evaluation metrics for real and synthetic data.

Sun *et al.* [23] have described a Gaussian matrix to model correlations between the weights of a neural network and to perform training and test for it. In addition to real data, they produced and used synthetic data. Kang *et al.* [24] have made a similar work using synthetic data to generate tasks to perform tests with multi-tasking learning. Ma *et al.* [25] have presented a trained artificial neural network with both real data and synthetic data, and they highlighted the synthetic data allowed a better investigation of the robustness of the model concerning its initialization and the randomness of the data. However, the generations of synthetic data performed in these last works are specific to their respective problems, and it is not possible a priori to reuse the same data for different applications.

[1] http://labvis.ufpa.br/datagen

There are few works that generate synthetic data facilitating the manipulation of data characteristics as well as creating complex patterns of data, without the need for programming skills, aiming at testing tools or machine learning algorithms or visualizing the information with a general context.

III. PROPOSED APPLICATION

The main goal of this work is to present a data generator application to assist researchers in testing information visualization techniques and machine learning algorithms. The tool allows manipulation of statistical generators to produce synthetic data according to users specification. Thus, the sharing of synthetic datasets can be done through a lightweight descriptive file that other researchers can use to re-create dataset profiles, easing replication of studies. Figure 1 shows a flow chart of the tool usage.

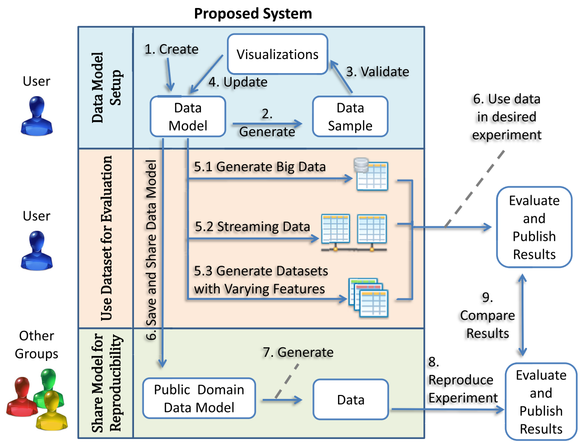


FIGURE 1. A typical use flow of the application.

The typical flow of the application consists of three phases: setup, use, and sharing. The creation of a synthetic dataset is an iterative process that starts when the user creates a data model (1). The creation of a data model is the specification and composition of generators to create a description of data behavior (e.g., a correlation between certain dimensions, or the presence of outliers). After any changes in the generators, the application produces a small-size sample dataset (2) for visual feedback of data behavior (3), which allows users to evaluate the used generators and update them if needed (4). It is important to highlight that the data that will be generated in the final file are not the same of the preview: the purpose of the preview is only to quickly show the behavior of the generators that the model will use to produce the final dataset.

After the setup step, users can decide to generate the data to test their applications, or to share the constructed data model for experiment replication. Aside from generating a dataset file that follow the generators of the model, users have a few extra options to create data: generating large volumes of data if necessary (5.1), feeding the tested technique or algorithm through a data generation streaming (5.2), and creating a set of similar datasets with slight differences in its features (5.3).

In this typical data flow, a user could choose to share the model with fellow researchers to reproduce experiments (6). The researchers receiving this data model could generate their own dataset following the same distribution defined by the generators (7). While the underlying randomness of the generation process implies that two datasets created from the same model are not identical, the data points are equivalent as they share the same characteristics and behavior (e.g., same correlations, probabilities, outliers). This way, researchers can easily reproduce experiments (8) even if they are working with massive synthetic datasets, leading to a quick comparison of results (9).

A. SYSTEM ARCHITECTURE OVERVIEW

Figure 2 shows an overview of the application architecture: the gray boxes represent the main components, and the blue boxes represent the output type of the generated data. The output can either be a lightweight description of the model, a file with control data points, or a visualization of the data.

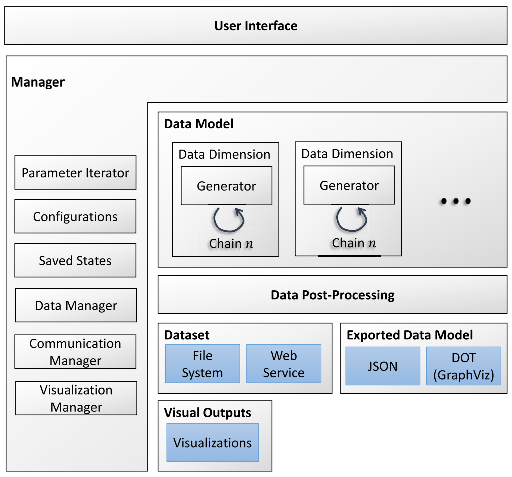


FIGURE 2. An overview of the application architecture.

1) MANAGER

The manager module receives requests through the graphical user interface, and it is responsible for forwarding those requests to its submodules. It is a cross-cutting module that coordinates the flow of information through the whole application, intermediating the user interface with the generation logic. The submodules are:

- Parameter Iterator:
- Configurations:
- Saved States:
- Data Manager:
- Communication Manager:
- Visualization Manager:

2) DATA MODEL

The data model has the responsibility of handling the data dimensions of the dataset and the generators that produce the values. The data model is a representation of a dataset, composed of specifications that describe data behavior. The data models can generate not only the final complete dataset but also data samples, which are small-size datasets (default size of 100 rows) that have the same specified behavior. The data samples enable visual feedback, as they can be quickly visualized to validate if its characteristics match the testing requirements. It is important to highlight that when using the same data model to create more than one dataset (or to create data samples), the resulting data are similar (i.e., has the same behavior), but not necessarily have the same data values.

It is possible to export the data model and share it with fellow researchers, which eases the reproduction of an experiment since the exported data model is often lighter than a massive dataset, thus being easier to store and download.

3) DIMENSIONS

The data model is composed of dimensions, each containing a chain of generators. The dimensions have the responsibility of holding the rules that drive data generation. Each dimension has four elements: order number, title, data type, and generator chain. The data type of the dimension depends on the generation rules associated with them and can be numerical, categorical, time, or mixed.

4) GENERATORS

The generators are responsible for creating and modifying values. Several generators can sequentially compose a generator chain, implemented with the Decorator design pattern [26]. Every generator has a reference to its parent and to its child, so the communication through the chain can be bilateral. The results of a generator depends on the result of its children, simulating a cascade system with the data returned by each generator.

Figure 3 shows a general scheme of the generator chain, with its inputs and outputs. For each generator in the chain, the user defines parameters and an operator $(\cdot)$. The parameters are the arguments that generators need in order to produce values (e.g., mean μ and standard deviation σ in Gaussian generators). The operator combines the value returned by one generator with the value returned by its child, and it can be sum, subtraction, multiplication, division, and modulo.

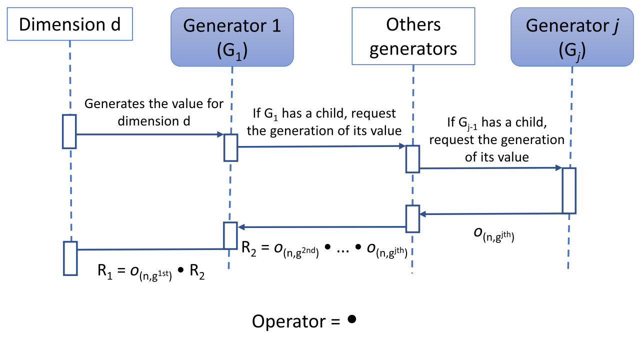

Dimension d
Generator 1 (G$_1$)
Others generators
Generator j (G$_j$)
Generates the value for dimension d
If G$_1$ has a child, request the generation of its value
If G$_{j-1}$ has a child, request the generation of its value
$o_{(n,g^{jth})}$
R$_2$ = $o_{(n,g^{2nd})}$ • … • $o_{(n,g^{jth})}$
R$_1$ = $o_{(n,g^{1st})}$ • R$_2$
Operator = •

FIGURE 3. A general scheme of the generator chain for one dimension *d*.

Consider that a dataset DS is a set of n entries $DS = \{E_1, E_2, \ldots, E_n\}$, where each entry $E_{i|1 \leqslant i \leqslant n}$ is a set of m values $E_i = \{v_{(i,1)}, v_{(i,2)}, \ldots, v_{(i,m)}\}$, and each value $v_{(i,j)|1 \leqslant j \leqslant m}$ is related to a data dimension. Besides, each data dimension has a chain of generators $G_j = \{g_{(j,1)}, g_{(j,2)}, \ldots, g_{(j,\omega)}\}$, which is responsible to generate the values $\{v_{(1,j)}, v_{(2,j)}, \ldots, v_{(n,j)}\}$.

In order to create a value $v_{(i,j)}$, the generators recursively operate their results as follows:

$$\begin{aligned} r_k &= g_{(j,k)} \cdot r_{k-1}, \\ r_1 &= g_{(j,1)} \end{aligned}$$

Being r_1 the initial step, and $r_\omega = v_{(i,j)}$, which is the result of the recursion when it reaches the last generator $g_{(j,\omega)}$.

The current version of the application has 36 different generators; each one has a unique behavior to generate data, which may change depending on its child generator. Hence, each generator in the chain is a building block to design a customized data distribution. There are five major types of generators: Random, Geometric, Accessory, Function, and Sequence.

a: THE RANDOM GENERATORS

produce each new value independently and randomly following a probability density or rule. For example, the Gaussian generator creates values based on a predefined mean μ and standard deviation σ, while the Uniform generator creates values between a minimum *min* and maximum *max* values with the same probability to any value in the range. Random generators can be composed using the user-defined operations to create new distributions (e.g., a uniform distribution might be summed up with a Gaussian distribution).

Table 1 shows the list of Random generators currently available in the application. The params are constants that can be of type real $\mathbb{R}$ or categorical C. The output column shows illustrations of the probability density functions that drives value generation.

b: THE GEOMETRIC GENERATORS

create numerical data following geometrical primitives. The user specifies parameters of shapes in a space $\mathbb{R}^2$, and the generator produces data points following the specified pattern.

Since geometrical shapes are specified on the space $\mathbb{R}^2$, a single data dimension can not represent the values: the output is not a value o_n, but actually an ordered pair (o_n1, o_n2). In order to generate such 2-dimensional information, an extra data dimension is also needed.

When associating a Geometric generator to the generator chain of a certain dimension, the generator only returns the first element o_{n1} of the ordered pair. If users want to generate the other element o_{n2} of the pair, they need to add a new dimension to the data model and use a particular generator called Get Extra; the subsection on Accessory generators details the behavior of the particular generator.

Table 2 shows the list of Geometric generators. The params are constants $(a1, a2, a3)$ of type real $\mathbb{R}$ that describe the

TABLE 1. The list of Random generators.

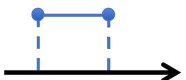
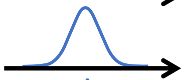
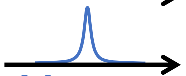
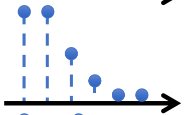
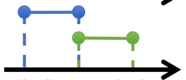
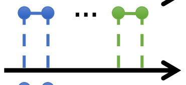
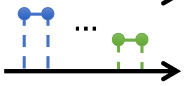
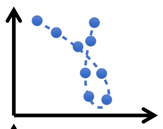
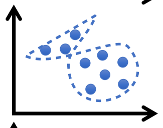
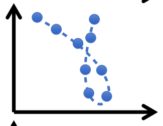
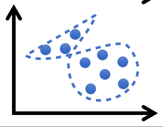
Name	Params.	Output
Uniform	$(min, max) \in \mathbb{R}$	
Gaussian	$(\mu, \sigma) \in \mathbb{R}$	
Cauchy	$(x_0, \gamma) \in \mathbb{R}$	
Poisson	$\lambda \in \mathbb{R}$	
Bernoulli	$p \in \mathbb{R}$	
Categorical	$a_{1,\ldots,z} \in C$	
Weighted Categorical	$a_{1,\ldots,z} \in C$, $b_{1,\ldots,z} \in \mathbb{R}$	

TABLE 2. The list of Geometric generators.

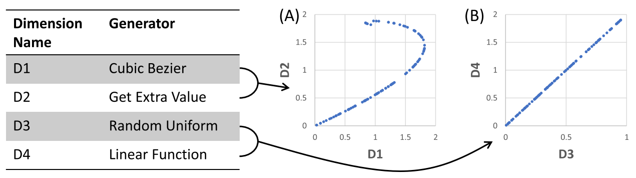
Name	Params.	Output
Stroke Quadratic Bézier	$a_{1,2,3} \in \mathbb{R}^2$	
Fill Quadratic Bézier	$a_{1,2,3} \in \mathbb{R}^2$	
Stroke Cubic Bézier	$a_{1,2,3,4} \in \mathbb{R}^2$	
Fill Cubic Bézier	$a_{1,2,3,4} \in \mathbb{R}^2$	

behavior of the shapes, such as control points. The output column illustrates how each generator distributes data points through the specified shape.

The Figure 4 illustrates how to use the geometric generators. The Cubic Bezier Stroke generator is assigned to dimension D1, which corresponds to the first element o_{n1} of the pair. A Get Extra Accessory then gets the second element o_{n2} and associates it with dimension D2. The user can create different chains for each dimension, for instance, adding noise only to dimension D2.

c: THE ACCESSORY GENERATORS

are responsible for modifying the values returned by other generators. For example, the Missing Value Accessory generator is responsible for randomly transforming the values



Dimension Name	Generator
D1	Cubic Bezier
D2	Get Extra Value
D3	Random Uniform
D4	Linear Function

FIGURE 4. Using the Geometric generators and Get Extra Acessory.

generated from the child generator into missing values; the user may choose the percentage amount of missing values. The output o_n can be deterministic (e.g., MinMax, Linear Scale), or probabilistic (e.g., Constant Noises, Missing Values).

Since generators can only create a single value at a time, generators are not able to return data in the format of a n-tuple $(a_1, a_2, \ldots, a_n)$, thus returning only the first element $a1$. In order to access the other elements, a Get Extra Accessory enables retrieving a specific element from a returned tuple. Consequently, to access all values of an n-tuple generator, $n - 1$ extra dimensions must be created, each one with a Get Extra Accessory.

Table 3 shows all Accessory generators and their respective parameters and outputs. The params are constants (a_1, a_2, a_3) that can be of type real $\mathbb{R}$, probability P, or natural $\mathbb{N}$. In the case of random noise, an additional parameter is the probability distribution $r(\alpha)$ of the noise (e.g., Gaussian or uniform). If an accessory receives a value that does not fit its constraints (e.g., Range Filter receives a value outside the range) the value is invalidated, and the accessory makes another call to the child $ch(A)$ to obtain a new value.

TABLE 3. The list of Accessory generators.

Name	Params.	Output
Missing Value	$a \in \mathbb{P}$	$P(o_n = i_n) = 1 - a$ and $P(o_n = \varnothing) = a$
Range Filter	$a_{1,2} \in \mathbb{R}$	$o_n = \begin{cases} i_n & a_1 \geq i_n \geq a_2 \\ ch(A) & a_1 < i_n < a_2 \end{cases}$
Linear Scale	$a_{1,2,3,4} \in \mathbb{R}$	$o_n = \frac{i_n - a_1}{a_2 - a_1}(a_4 - a_3) + a_3$
MinMax	$a_{1,2} \in \mathbb{R}$	$o_n = \begin{cases} i_n & a_1 < i_n < a_2 \\ a_1 & i_n \leq a_1 \\ a_2 & i_n \geq a_2 \end{cases}$
Random Noise	$a_1 \in \mathbb{P}$, $a_2 \in \mathbb{R}$, $r(\alpha)$	$P(o_n = i_n) = 1 - a_1$ and $P(o_n = i_n + a_2 r(\alpha)) = a_1$
Constant Noise	$a_1 \in \mathbb{P}$, $a_2 \in \mathbb{R}$	$P(o_n = i_n) = 1 - a_1$ and $P(o_n = i_n + a_2) = a_1$
Low Pass Filter	$a \in \mathbb{R}$	$o_n = \frac{i_n - i_{n-1}}{a} + i_n$
No Repeat	-	$o_n = \begin{cases} i_n & i_n \notin \{i_{n-1}, \ldots, i_1\} \\ ch(A) & i_n \in \{i_{n-1}, \ldots, i_1\} \end{cases}$
Get Extra	$a \in \mathbb{N}$	$o_n = a^{th}(i_n)$

d: THE FUNCTION GENERATORS

transform the values generated in a previous dimension into new ones, enabling the creation of correlated dimensions. To use a Function generator, the user need to specify the dimension d that provides the values to be transformed (i.e., the domain of the function). Function generators can be linear, logarithmic, exponential, sinusoidal, quadratic, polynomial, categorical, numerical piecewise, and time piecewise. For instance, the user can make one dimension be inversely correlated to another by using a linear function generator that has a negative slope.

Table 4 shows the list of Function generators currently available in the application. The params can be of type real $\mathbb{R}$, or time T. Function generators modify values generated in another dimension d, so the output o_n always depends on the value d_n. Hence, the value d_n coming from another dimension is the domain of the function, and the output o_n is the image.

TABLE 4. The list of function generators.

Name	Params.	Output
Linear	$a_{1,2} \in \mathbb{R}$	$o_n = a_1 d_n + a_2$
Quadratic	$a_{1,2,3} \in \mathbb{R}$	$o_n = a_1 d_n^2 + a_2 d_n + a_3$
Polynomial	$a_{1,\ldots,z} \in \mathbb{R}$	$o_n = \sum_{k=0}^{z-1} a_{k+1} d_n^k$
Exponential	$a_{1,2} \in \mathbb{R}$	$o_n = a_1^{d_n} a_2$
Logarithm	$a_{1,2} \in \mathbb{R}$	$o_n = \log_b(d_n)$
Sinusoidal	$a_{1,2,3} \in \mathbb{R}$	$o_n = a_1 \sin(a_2 d_n + a_3)$
Categorical	-	$o_n = \begin{cases} ch_1(A) & d_n = 1^{th}(C) \\ \vdots & \vdots \\ ch_z(A) & d_n = z^{th}(C) \end{cases}$
Piecewise Time	$a_{1,\ldots,z} \in T$	$o_n = \begin{cases} ch_1(A) & d_n < a_1 \\ \vdots & \vdots \\ ch_z(A) & a_{z-1} \leq d_n < a_z \\ ch_{z+1}(A) & a_z \leq d_n \end{cases}$
Piecewise	$a_{1,\ldots,z} \in \mathbb{R}$	$o_n = \begin{cases} ch_1(A) & d_n < a_1 \\ \vdots & \vdots \\ ch_z(A) & a_{z-1} \leq d_n < a_z \\ ch_{z+1}(A) & a_z \leq d_n \end{cases}$

The Categorical, Piecewise Time, and Piecewise generators act as a switch-case function, where each case has a particular chain of generators. Thus, being z the number of cases in the switch-case, the Function generator ramifies the chain into z children $ch_1, ch_2, \ldots, ch_z$. The children used to generate the output o_n depends on the value d_n of another dimension.

e: THE SEQUENCE GENERATORS

create values according to an algorithm guided by parameters $(a_1, a_2, \ldots, a_z)$, the data index (n), and the previous value (o_{n-1}). The sequences can be arithmetic, geometric, or recursive and can have characteristics such as: being an increasing or decreasing sequence, have convergent values, or bounded ranges.

TABLE 5. The list of Sequence generators.

Name	Params.	Output
Constant	$a \in \mathbb{M}$	$o_n = a_1$
Counter	$a_{1,2} \in \mathbb{R}$	$o_n = o_{n-1} + a_2 \mid o_1 = a_1$
Time	$a_{1,2} \in \mathbb{T}$	$o_n = o_{n-1} + a_2 \mid o_1 = a_1$
Poisson Time	$a_{1,2} \in \mathbb{T},\ \lambda \in \mathbb{R}$	$o_n = o_{n-1} + \frac{a_2}{poisson(\lambda)} \mid o_1 = a_1$
Sinusoidal	$a_{1,2,3,4,5} \in \mathbb{R}$	$o_n = a_1 \sin(a_2 c_n + a_3) \mid c_n = c_{n-1} + a_5$ and $c_1 = a_4$
Custom	-	$o_n =$ user defined
Categorical	$a_{1,\ldots,z} \in C$, $b_{1,\ldots,z-1} \in \mathbb{N}$	$o_n = a \begin{cases} 1 & n \leq \sum_{k=1}^{1} b_k \\ \vdots & \vdots \\ z-1 & n \leq \sum_{k=1}^{z-1} b_k \\ z & n > \sum_{k=1}^{z-1} b_k \end{cases}$

Table 5 shows the list of Sequence generators currently available in the application. The params can be of type mixed M, real $\mathbb{R}$, time T, categorical C, or natural $\mathbb{N}$. Additionally, the Poisson Time Sequence Generator requires the parameter λ of the Poisson distribution.

When a sequence requires a previous output o_{n-1}, it needs an initial step for the first value generated o_1. The Sinusoidal generator depends on previous angles c_n instead of outputs, so the initial value is given by c_1.

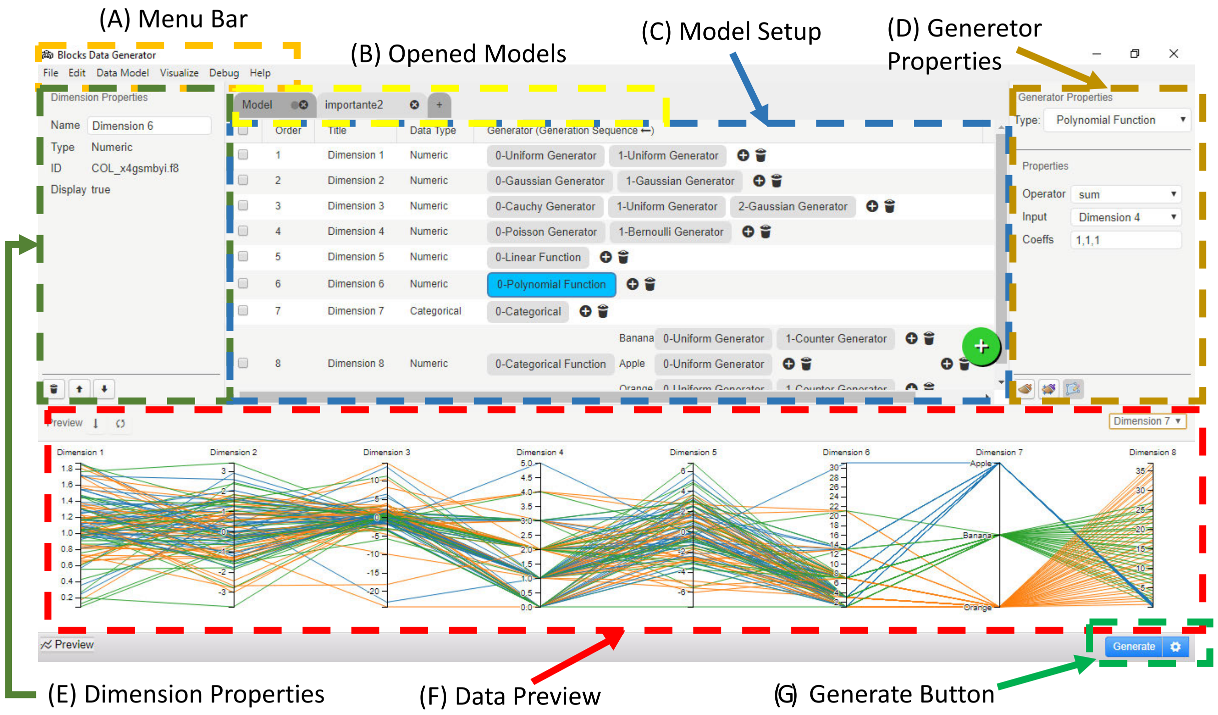
In the Custom Sequence generator, the user defines the custom sequence logic through a textual rule that specifies the values of each o_n using arithmetic operations (e.g., sum, multiplication, subtraction, and division), the previous value $x = o_{n-1}$ and data index n. For instance, the user can create a counter sequence typing 'n' as the text rule, so the values are equal to the index.

5) OUTPUT DATA

After finishing the data model, the user has a few options on how to export it: export data points, export data model specification, stream data through web service, and export a data model diagram.

If the data model is ready to create the final dataset, users can start the generation process to save data points into the file system. Alternatively, the system can generate a stream of data through a Web Service, in which data is generated and served upon URL (Uniform Resource Locator) requests.

If the user wants to export only the data model instead of the complete dataset, the system can generate a JSON (JavaScript Object Notation) file of the model. This JSON file saves the whole model in a lightweight hierarchical structure that preserves generators, operators, and parameters, so it can be later imported to the system to restore the data model. Another way to export the data model is through a DOT



Dimension Properties

Name	Dimension 6
Type	Numeric
ID	COL_x4gsmbyi.f8
Display	true

Order	Title	Data Type	Generator (Generation Sequence ⟵)
1	Dimension 1	Numeric	0-Uniform Generator, 1-Uniform Generator
2	Dimension 2	Numeric	0-Gaussian Generator, 1-Gaussian Generator
3	Dimension 3	Numeric	0-Cauchy Generator, 1-Uniform Generator, 2-Gaussian Generator
4	Dimension 4	Numeric	0-Poisson Generator, 1-Bernoulli Generator
5	Dimension 5	Numeric	0-Linear Function
6	Dimension 6	Numeric	0-Polynomial Function
7	Dimension 7	Categorical	0-Categorical
8	Dimension 8	Numeric	0-Categorical Function; Banana: 0-Uniform Generator, 1-Counter Generator; Apple: 0-Uniform Generator; Orange: 0-Uniform Generator, 1-Counter Generator

Generator Properties

Type: Polynomial Function

Properties

Operator	sum
Input	Dimension 4
Coeffs	1,1,1

FIGURE 5. The main graphical interface of the application.

file, which can be loaded into GraphViz [27] to produce a human-readable diagram of the model.

B. USER INTERFACE

Figure 5 shows the graphical user interface highlighting seven parts: Menu Bar (A), Opened Models Tabs (B), Model Setup Panel (C), Generator Properties (D), Dimension Properties (E), Data Preview (F), and Generate Button (G).

Figure 5 (A) shows the application's menu bar with the menus *File*, *Edit*, *Data Model*, *Visualize*, and *Help*.

The *File* menu presents the functions *New model*, *New dimension*, *Open model*, *Save model*, *Save model as*, and *Import Dataset*. The *Edit* menu has two options *Undo* and *Redo*. In the *Visualize* menu, the user can choose from several visualization techniques to see the data samples of the current model; the currently implemented techniques are [28], [29]: bar chart, histogram, scatterplot matrix, beeswarm plot, treemap, sunburst, parallel coordinates, and bundled parallel coordinates [30]. The *Model* menu has the options *Rename*, *Delete*, *Export.DOT File*, *Copy Model ID*, *Copy URI Web Service*, *Toggle Web Service*, *Open Web Service*.

Figure 5 (B) shows the tabs of opened models. Each tab contains a data model specification in a setup panel (C), that includes the title and data type of dimensions, the generator chains, and utility buttons such as filter, add generator, delete generator, and delete dimension. Users can add a new dimension to the model through the + button at the bottom-right of the panel. Additionally, it is also possible to add, remove, and change the position of generators. The user can also filter out dimensions, so they are omitted from both data preview and final dataset. When users select a generator, information is displayed about its associated dimension (E) and its own properties (D).

The generator properties panel (D) is where the user defines the generator type, and inputs its parameters and operators. After any changes in the model, the Data Preview panel (F) updates a parallel coordinates visualization that shows the data samples, allowing for quick visual feedback of data behavior.

The blue button "generate" (G) at the bottom-right of the window opens the dialog for creating the final dataset, in which users configure name, path, and number of lines. Clicking the gear button displays a new window to configure the Parameter Iterator which generates a sequence of datasets varying some parameters iteratively.

Besides the Preview Panel, the system has a built-in visualization analysis tool to show the data samples of the model *(Menu > Visualize)*. This feature becomes essential because the user can visually verify in real time if the data model is generating data according to the expectations.

In the visualization window—which can be a different window from the main one—users can choose the visualizations they prefer to use. Figure 6 shows that the visualization window has a flexible layout, allowing the user to resize each visualization by dragging the dotted line, as well as split an area to add new ones. The visualizations are coordinated through the colors, filters, and selections, so it is easier to relate data items from different views.

Also, the user can open more than one window at a time to see them on different screens if needed.

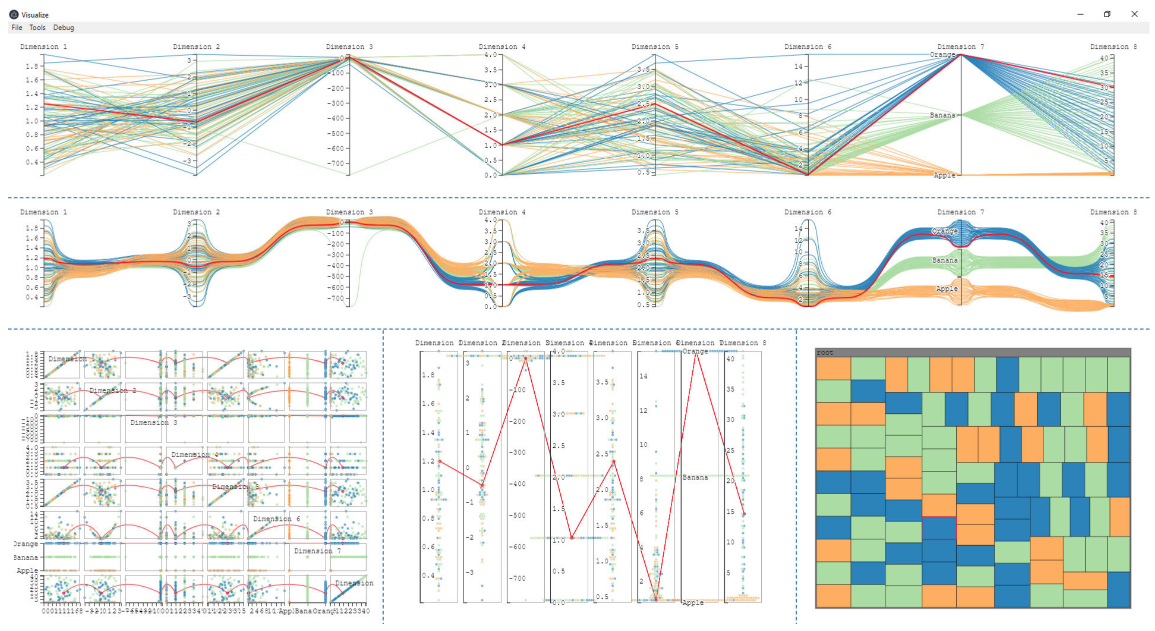


FIGURE 6. The visualization window shows sample data of the model, the red lines show the coordinated brushing between them.

IV. USAGE SCENARIOS - GENERATING DATASETS FOR MACHINE LEARNING CLASSIFICATION

This scenario is an example of how to generate a test dataset for machine learning with variation in specific data characteristics. To this end, the proposed tool will generate datasets varying the number of outliers, class separation, amount of missing values, class imbalance, amount of bad features, and amount of classes [31]–[37].

The variations are specified on top of a default dataset, which has the following characteristics:

- 1.000 entries
- No outliers
- No missing values
- Two dimensions (one relevant feature and one class, no bad features)
- 80% Class separation
- Two Classes
- No Class Imbalance

Figure 7 shows how the system generates this default dataset. A Categorical Function act as the switch-case that correlates the class dimension (Dimension 1) with the feature dimension (Dimension 2). The chains in Dimension 2 contain a Uniform generator whose parameters depend on the value of Dimension 1: for class A1 the parameters are $Min = 0$ and $Max = 1.2$, and for class A2 they are $Min = 1$ and $Max = 2$. These parameters create an 20% overlap in the feature dimension, so only 80% of the classes are separated.

Thus, six types of datasets were generated, one for each of the six characteristics in the default dataset. In each type of dataset, the system generated four datasets with slight differences in the associated characteristic. For instance, to vary the effect of the number of outliers, the system created datasets

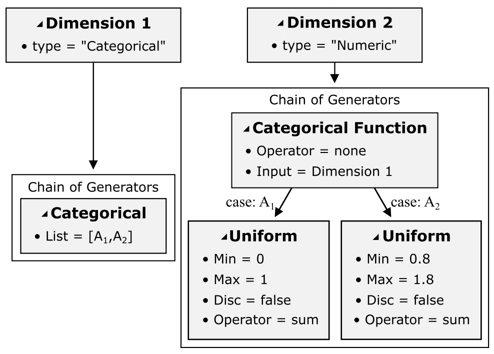


FIGURE 7. The generators that builds the default dataset.

with 10%, 20%, 30%, and 40% of outliers, without changing the other characteristics. The variations of the characteristics are the following:

- Amount of outliers: [10%, 20%, 30%, 40%]
- Class separation: [90%, 80%, 70%, 60%]
- Amount of missing values: [10%, 20%, 30%, 40%]
- Class imbalance: [50%-50%, 40%-60%, 30%-70%, 20%-80%]
- Bad features: [1-1, 1-3, 1-5, 1-7]
- Amount of classes: [2, 12, 22, 32]

A. AMOUNT OF OUTLIERS

The Amount of Outliers is the proportion of outliers in the data. Figure 8 shows how the Noise Generator can be used to produce the outliers. The noise generator was configured to change the original value adding it with a gaussian noise (mean 0 and standard deviation 1) multiplied by 20 (force parameter) and occurring in a certain percentage (varied from 10% to 40%) using the uniform distribution.

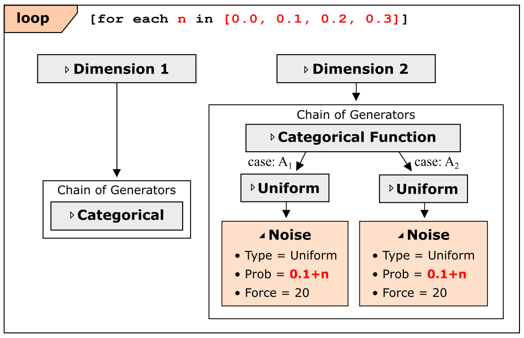


FIGURE 8. **Adding noise generators to the uniform distributions creates outliers.**

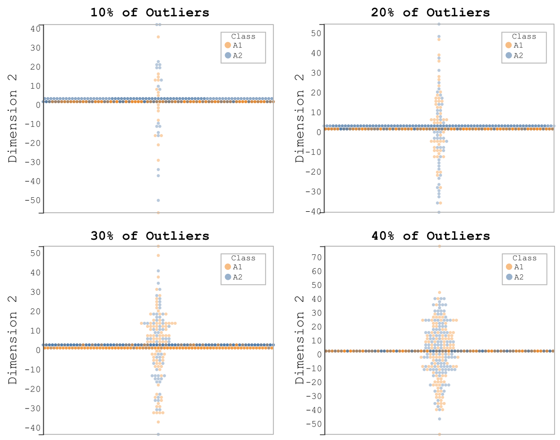


FIGURE 9. **The accuracy of the models with varying amount of outliers.**

Figure 9 shows the generated datasets varying the number of outliers. The beeswarm plot shows the majority of the data around 0 and 1.8 and some outliers above and below. By visualizing the sequence of charts from 10% to 40%, it is evident that as the 'Prob' parameter (see Figure 8 in box Noise) increases, the number of outliers increases.

B. CLASS SEPARATION

The Class Separation characteristic refers to the amount of overlap in the distributions of each class. Figure 10 shows how the parameters ('Min' and 'Max') of the Uniform generators can be changed to create an overlap between distributions. For instance, to create a class separation

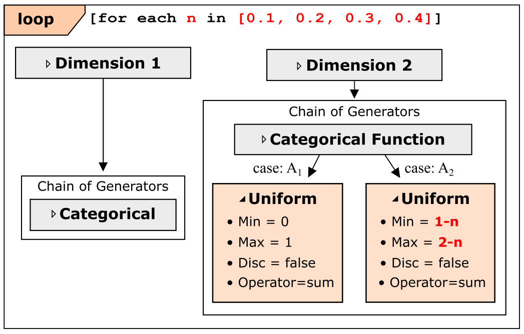


FIGURE 10. **Sliding the interval of the uniform distribution of one class increases or decreases the class separation.**

of 60%, 40% of the dimension range should be shared by the Uniform Generators (e.g., C1: $Min = 0$ and $Max = 1.4$; C2: $Min = 0.6$ and $Max = 2$; the interval [0.6, 1.4] is shared by both generators).

Figure 11 shows the amount of separation between classes. The Histogram presents accumulated value for each class of the dataset, with a clear separation in the 0.96 mark for 90% separation. From there, the overlap of the distributions of each class increases. These datasets could be used to test how the accuracy of classifiers decreases as the overlap between classes increases.

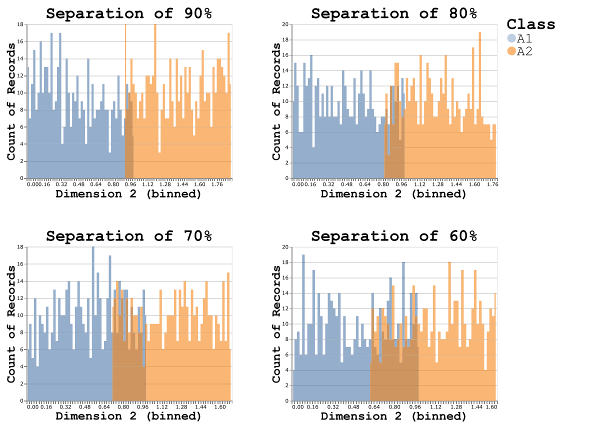


FIGURE 11. **The distribution of values with varying class separation.**

C. AMOUNT OF MISSING VALUES

The amount of missing values is the proportion of empty values in the data. Figure 12 shows how the MCAR (Missing Completely at Random) generator produces this characteristic. This generator gets the value generated by another generator (in this case, a Uniform generator) and changes it to a missing value according to a probability, the parameter 'Prob.'

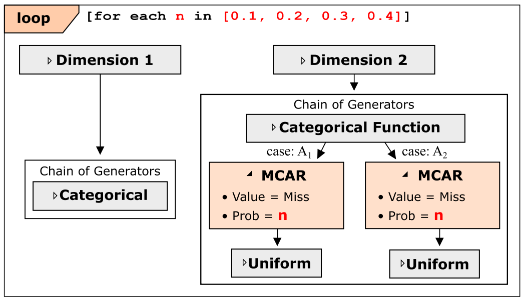


FIGURE 12. **Adding an MCAR accessory before the uniform distributions create missing values randomly according to a probability.**

Figure 13 presents the missing values of datasets. The red color is used to map the missing values, and the ratio of change from 10% to 40% is shown in Dimension 2 as the red increases, as the class remains unchanged, being the desired scenario to evaluate the classification approaches with missing values. An opacity value of 0.2 is used on the red color to not clutter the visualizations.

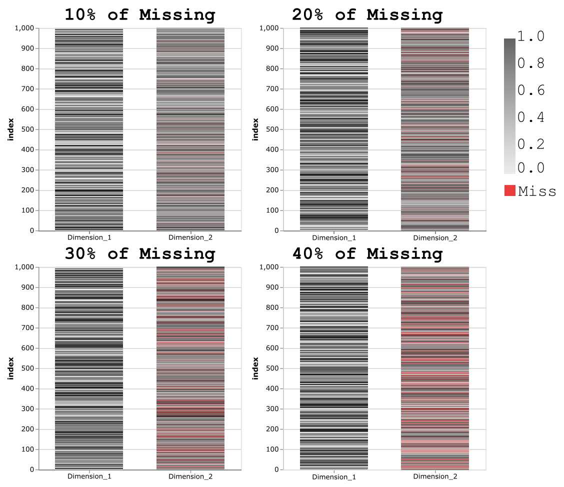


FIGURE 13. The amounts of missing value by its probability.

The datasets generated in this case could be used to test the robustness of a classifier when there are missing values in the features. Other types of missing values could be generated by the presented system, such as the MAR (Missing at Random). It could also be used to test the imputation algorithms.

D. CLASS IMBALANCE

The Class Imbalance is the proportion of each class in the dataset. The class imbalance presents a list of challenges that could be tested with new approaches in classification [38]. Figure 14 shows how the Weighted Categorical generator produces this characteristic, with the weight (or likelihood) of each category being the proportion it appears in the dataset.

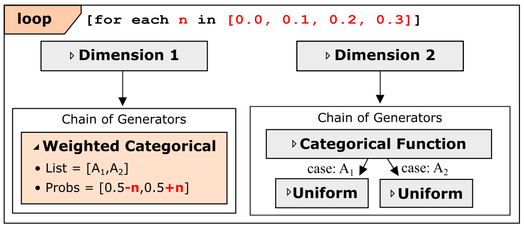


FIGURE 14. Changing the categorical generator to weighted categorical allows the generation of class-imbalanced datasets.

Figure 15 shows the class imbalance on beeswarm plots. On the balanced plot, the thickness of both plots are equal, but as the imbalance starts, the thickness of both distributions start to change. At the end (20%-80%) the thickness of A1 is drastically reduced, and the thickness of A2 grew.

E. BAD FEATURES

The Bad Features characteristic refers to the number of features that are unrelated to the class, e.g., there is no correlation between the class and the feature, being 'bad' for classification. Figure 16 show that adding dimensions with Uniform Generators is enough, as the dimensions are unrelated by default.

Figure 17 shows the bad features on a Parallel Coordinates. The visual distinction of the good feature presents is clear, and

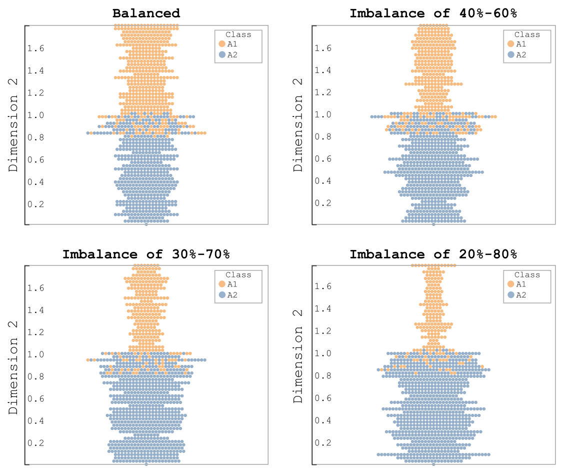


FIGURE 15. Imbalance of classes on beeswarm plots.

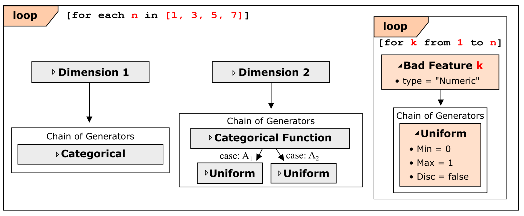


FIGURE 16. Adding dimensions without categorical functions creates bad features unrelated to the class.

the values of the bad features are shown with clutter and irrelevant patterns. These datasets could be used to test whether a classifier well separates the good features of useless features.

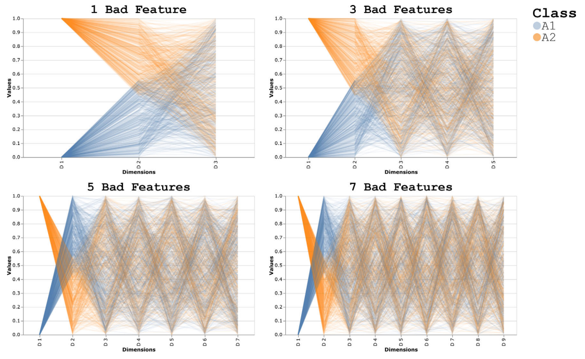


FIGURE 17. Parallel Coordinates showing bad features.

F. AMOUNT OF CLASSES

The amount of classes refers to the number of different categories in the class dimension. Figure 18 shows how the Categorical Generator can accept many categories as parameters, and how they impact the switch case (the Categorical Function) in the feature dimension.

Figure 19 shows the number of classes on a beeswarm plot binned by class. The color labels each class along with the position. The size of the class plots gets very small from 22 classes onward, but the distribution of data presents the separation of different classes. These datasets could be used to test which classifiers are robust when classifying many

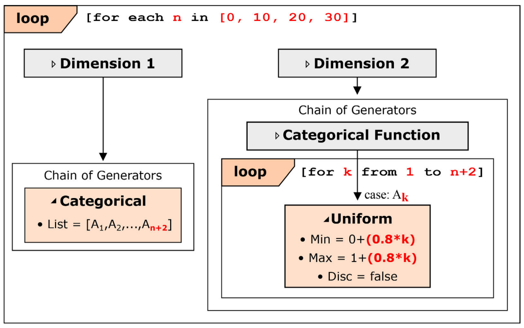


FIGURE 18. Adding categories to the Categorical Generator increases the number of classes.

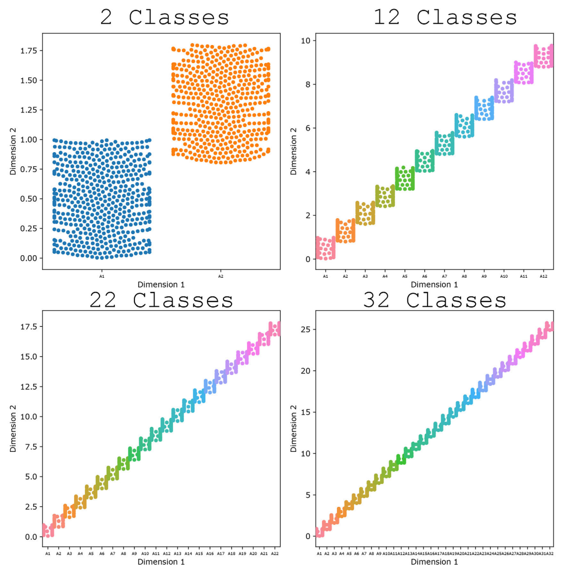


FIGURE 19. The accuracy of the models with varying amount of classes.

classes, also test if the accuracy of classifiers remains balanced between classes.

V. CONCLUSION

This work presented a synthetic data generator for the evaluation of information visualization techniques and machine learning systems. The application is flexible and gives the user the freedom to create custom generation profiles by composing several data distribution primitives, such as uniform and normal distributions. It also contains accessories, functions, sequences, and geometric generators that allow highly customizable datasets.

The descriptive model file can be exported and imported, favoring the reproducibility of research tests and experiments. The application also offers a data stream web service to ease the interoperability of the system and its generated data in external applications.

This article also presents how the application can be used in the context of evaluating machine learning algorithms. It showed how different datasets could be generated, allowing control of the system over common problems on machine learning tasks. The visualizations built for each scenario show the consistency of the tool on validating each situation. The datasets created for this article are freely available in the IEEE DataPort under the following link: http://dx.doi.org/10.21227/5aeq-rr34.

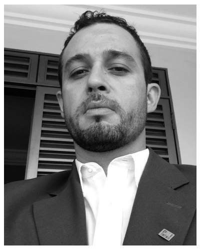
As future works, the extraction of generators from real data can be explored. The modeling language that describes how to compose a chain of generators can be used as an idiom to understand how real data behaves. Breaking through this problem would open possibilities to edit real data, changing the parameters that drive their distributions to create synthetic data from real ones. Another option is the use of machine learning techniques to make the generated synthetic data more realistic by adding noises that are common in real data without severely changing the underlying distribution.

Additionally, the authors intend to add: more types of generators, new ways to display generators to facilitate the understanding of the model's design, a constant seed to enable the generation of a unique dataset, new visualizations for data validation, and new interaction possibilities, such as zoom in/out, filter, and re-ordering. Another extension to future work could add a verification or comparison component either visually or using a quantitative metric. Besides that, another increment to the system will be the usage of real-world data to generate new synthetic data with corresponding characteristics where the user could compare both each other.

REFERENCES

[1] B. S. Santos and P. Dias, "Evaluation in visualization: Some issues and best practices," *Vis. Data Anal.*, vol. 9017, Feb. 2013, Art. no. 90170O. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2038259
[2] B. S. Santos, "Evaluating visualization techniques and tools: What are the main issues?" in *Proc. Workshop Beyond Time Errors Novel Eval. Methods Vis. (BELIV)*, 2008, pp. 1–2.
[3] R. Redpath and B. Srinivasan, "Criteria for a comparative study of visualization techniques in data mining," in *Intelligent Systems Design and Applications*. Berlin, Germany: Springer, 2003, pp. 609–620.
[4] H. Lam, E. Bertini, P. Isenberg, C. Plaisant, and S. Carpendale, "Empirical studies in information visualization: Seven scenarios," *IEEE Trans. Vis. Comput. Graphics*, vol. 18, no. 9, pp. 1520–1536, Sep. 2012.
[5] S. Liu, W. Cui, Y. Wu, and M. Liu, "A survey on information visualization: Recent advances and challenges," *Vis. Comput.*, vol. 30, no. 12, pp. 1373–1393, Dec. 2014.
[6] Y. P. dos Santos Brito, C. G. R. dos Santos, S. de Paula Mendonca, T. D. Araujo, A. A. de Freitas, and B. S. Meiguins, "A prototype application to generate synthetic datasets for information visualization evaluations," in *Proc. 22nd Int. Conf. Inf. Vis. (IV)*, Jul. 2018, pp. 153–158.
[7] S. Popić, B. Pavković, I. Velikić, and N. Teslić, "Data generators: A short survey of techniques and use cases with focus on testing," in *Proc. IEEE 9th Int. Conf. Consum. Electron. (ICCE-Berlin)*, Sep. 2019, pp. 189–194.
[8] R. A. DeMilli and A. J. Offutt, "Constraint-based automatic test data generation," *IEEE Trans. Softw. Eng.*, vol. 17, no. 9, pp. 900–910, Sep. 1991.
[9] M. Mann, O. P. Sangwan, P. Tomar, and S. Singh, "Automatic goal-oriented test data generation using a genetic algorithm and simulated annealing," in *Proc. 6th Int. Conf.-Cloud Syst. Big Data Eng. (Confluence)*, Jan. 2016, pp. 83–87.
[10] S. Rani and B. Suri, "An approach for test data generation based on genetic algorithm and delete mutation operators," in *Proc. 2nd Int. Conf. Adv. Comput. Commun. Eng. (ICACCE)*, May 2015, pp. 714–718.
[11] G. Albuquerque, T. Lowe, and M. Magnor, "Synthetic generation of high-dimensional datasets," *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 12, pp. 2317–2324, Dec. 2011.
[12] B. Wang, P. Ruchikachorn, and K. Mueller, "SketchPadN-D: WYDIWYG sculpting and editing in high-dimensional space," *IEEE Trans. Vis. Comput. Graphics*, vol. 19, no. 12, pp. 2060–2069, Dec. 2013.

[13] B. C. Kwon, H. Kim, E. Wall, J. Choo, H. Park, and A. Endert, "AxiSketcher: Interactive nonlinear axis mapping of visualizations through user drawings," *IEEE Trans. Vis. Comput. Graphics*, vol. 23, no. 1, pp. 221–230, Jan. 2017.
[14] R. Liu, B. Fang, Y. Y. Tang, and P. P. K. Chan, "Synthetic data generator for classification rules learning," in *Proc. 7th Int. Conf. Cloud Comput. Big Data (CCBD)*, Nov. 2016, pp. 357–361.
[15] P. J. Lin, B. Samadi, A. Cipolone, D. R. Jeske, S. Cox, C. Rendón, D. Holt, and R. Xiao, "Development of a synthetic data set generator for building and testing information discovery systems," in *Proc. 3rd Int. Conf. Inf. Technol., New Gener. (ITNG)*, 2006, pp. 707–712.
[16] D. R. Jeske, P. J. Lin, C. Rendón, R. Xiao, and B. Samadi, "Synthetic data generation capabilties for testing data mining tools," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2007, pp. 1–6.
[17] M. Pasinato, C. E. Mello, M.-A. Aufaure, and G. Zimbrão, "Generating synthetic data for context-aware recommender systems," in *Proc. 1st BRICS Countries Congr. Comput. Intell. (BRICS-CCI)*, Sep. 2013, pp. 563–567.
[18] J. Dahmen and D. Cook, "SynSys: A synthetic data generation system for healthcare applications," *Sensors*, vol. 19, no. 5, p. 1181, 2019.
[19] D. García and M. Millán, "A prototype of synthetic data generator," in *Proc. 6th Colombian Comput. Congr. (CCC)*, May 2011, pp. 1–6.
[20] *Graph Generation With Prescribed Feature Constraints*, Soc. Ind. Appl. Math., Philadelphia, PA, USA, Apr. 2009.
[21] F. Brodkorb, M. Kopp, A. Kuijper, and T. Von Landesberger, "A modular rule-based visual interactive creation of tree-shaped geo-located networks," in *Proc. 12th Int. Conf. Signal-Image Technol. Internet-Based Syst. (SITIS)*, 2016, pp. 397–403.
[22] D. T. Kofinas, A. Spyropoulou, and C. S. Laspidou, "A methodology for synthetic household water consumption data generation," *Environ. Model. Softw.*, vol. 100, pp. 48–66, Feb. 2018.
[23] S. Sun, C. Chen, and L. Carin, "Learning structured weight uncertainty in Bayesian neural networks," *Proc. Mach. Learn. Res.*, vol. 54, pp. 1283–1292, Apr. 2017.
[24] Z. Kang, K. Grauman, and F. Sha, "Learning with whom to share in multi-task feature learning," in *Proc. Int. Conf. Mach. Learn.*, 2011, pp. 1–8.
[25] J. Ma, Z. Zhao, X. Yi, J. Chen, L. Hong, and E. H. Chi, "Modeling task relationships in multi-task learning with multi-gate mixture-of-experts," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, New York, NY, USA, Jul. 2018, pp. 1930–1939.
[26] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA, USA: Addison-Wesley, 1995.
[27] J. Ellson, E. Gansner, L. Koutsofios, N. C. Stephen, and G. Woodhull, "Graphviz—Open source graph drawing tools," in *Proc. Int. Symp. Graph Drawing*. Berlin, Germany: Springer, 2001, pp. 483–484.
[28] S. Few, *Now You See it: Simple Visualization Techniques for Quantitative Analysis*. El Dorado Hills, CA, USA: Analytics Press, 2009.
[29] R. Spence, *Information Visualization: An Introduction*. London, U.K.: Springer, 2014.
[30] R. S. A. Divino, C. G. R. Santos, and B. S. Meiguins, "A visual representation of clusters characteristics using edge bundling for parallel coordinates," in *Proc. 21st Int. Conf. Inf. Vis. (IV)*, Jul. 2017, pp. 90–95.
[31] J. M. Johnson and T. M. Khoshgoftaar, "Survey on deep learning with class imbalance," *J. Big Data*, vol. 6, no. 1, p. 27, Dec. 2019.
[32] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, "A survey of machine learning for big data processing," *EURASIP J. Adv. Signal Process.*, vol. 2016, no. 1, p. 67, Dec. 2016.
[33] A. Tajer, V. V. Veeravalli, and H. V. Poor, "Outlying sequence detection in large data sets: A data-driven approach," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 44–56, Sep. 2014.
[34] J. L. Leevy, T. M. Khoshgoftaar, R. A. Bauder, and N. Seliya, "A survey on addressing high-class imbalance in big data," *J. Big Data*, vol. 5, no. 1, p. 42, Dec. 2018.
[35] S. Das, S. Datta, and B. B. Chaudhuri, "Handling data irregularities in classification: Foundations, trends, and future challenges," *Pattern Recognit.*, vol. 81, pp. 674–693, Sep. 2018.
[36] A. Dal Pozzolo, O. Caelen, Y.-A. Le Borgne, S. Waterschoot, and G. Bontempi, "Learned lessons in credit card fraud detection from a practitioner perspective," *Expert Syst. Appl.*, vol. 41, no. 10, pp. 4915–4928, Aug. 2014.
[37] Q. Xiang, X. Dai, Y. Deng, C. He, J. Wang, J. Feng, and Z. Dai, "Missing value imputation for microarray gene expression data using histone acetylation information," *BMC Bioinf.*, vol. 9, no. 1, p. 252, Dec. 2008.
[38] A. Ali, S. M. Shamsuddin, and A. L. Ralescu, "Classification with class imbalance problem: A review," *Int. J. Adv. Soft Comput. Appl.*, vol. 7, no. 3, pp. 176–204, 2015.



SANDRO DE PAULA MENDONÇA received the master's degree from the Federal University of São Carlos. He is currently pursuing the Ph.D. degree with the Laboratory of Visualization, Interaction, and Intelligent Systems, Federal University of Pará (UFPA). His research topics are information data quality and visual analytics.

YVAN PEREIRA DOS SANTOS BRITO has been pursuing the master's degree in computer science with the Laboratory of Visualization, Interaction, and Intelligent Systems, Federal University of Pará (UFPA), since 2020. His research interest includes information visualization.

CARLOS GUSTAVO RESQUE DOS SANTOS received the master's and Ph.D. degrees from the Federal University of Pará, in 2015 and 2017, respectively. He is currently a Tenured Professor with Federal University of Pará. His research interests are in information and scientific visualization, virtual and augmented reality, and human computer interaction.

RODRIGO DO AMOR DIVINO LIMA is currently pursuing the master's degree with the Laboratory of Visualization, Interaction, and Intelligent Systems, Federal University of Pará (UFPA), where his researches are information visualization and visual analytics topics.

TIAGO DAVI OLIVEIRA DE ARAÚJO received the master's degree from the Laboratory of Visualization, Interaction, and Intelligent Systems, Federal University of Pará (UFPA), where he is currently pursuing the Ph.D. degree. His research interests are information and visualization, virtual and augmented reality, and computer vision.

BIANCHI SERIQUE MEIGUINS received the master's degree from the Pontifical Catholic University of Campinas, in 1999, and the Ph.D. degree from the Federal University of Pará, in 2003. He is currently a Tenured Professor with Federal University of Pará. His research interests are in information and scientific visualization, virtual and augmented reality, and human computer interaction.

• • •