# Approach to Synthetic Data Generation for Imbalanced Multi-class Problems with Heterogeneous Groups

Dennis Treder-Tschechlov[1] Peter Reimann[2] Holger Schwarz[1] Bernhard Mitschang[1]

**Abstract:** To benchmark novel classification algorithms, these algorithms should be evaluated on data with characteristics that also appear in real-world use cases. Important data characteristics that often lead to challenges for classification approaches are multi-class imbalance and heterogeneous groups. Heterogeneous groups are sets of real-world entities, where the classification patterns may vary among different groups and where the groups are typically imbalanced in the data. Real-world data that comprise these characteristics are usually not publicly available, e. g., because they constitute sensitive patient information or due to privacy concerns. Further, the manifestations of the characteristics cannot be controlled specifically on real-world data. A more rigorous approach is to synthetically generate data such that different manifestations of the characteristics can be controlled as well. However, existing data generators are not able to generate data that feature both data characteristics, i. e., multi-class imbalance and heterogeneous groups. In this paper, we propose an approach that fills this gap as it allows to synthetically generate data that exhibit both characteristics. We make use of a taxonomy model that organizes real-world entities in domain-specific heterogeneous groups to generate data reflecting the characteristics of these groups. Further, we incorporate probability distributions to reflect the imbalances of multiple classes and groups from real-world use cases. The evaluation shows that our approach can generate data that feature the data characteristics multi-class imbalance and heterogeneous groups and that it allows to control different manifestations of these characteristics.

**Keywords:** Machine Learning, Classification, Data Generation, Real-world Data Characteristics

## 1 Introduction

Data are the basis to evaluate and benchmark classification algorithms. For such benchmarks, algorithms should be evaluated on data that reflect characteristics that also appear in real-world use cases. Besides general characteristics, such as the number of data instances, features, or classes, we focus on characteristics that lead to significant challenges for classification algorithms and that are present in many real-world use cases. According to major literature in this field, two of the most challenging data characteristics are **multi-class imbalance** [HG09, Ga12, WY12] and **heterogeneous groups** [SWK09, HRM19, Me21, SG21]. Multi-class imbalance means that the data contains multiple classes that are unevenly distributed [HG09]. This leads to the challenge that less frequent classes are typically ignored by classification algorithms [WY12]. We define groups as specific sets of real-world entities, e. g., genders, ethnic groups [SG21], or product groups [HRM19]. These groups are usually heterogeneous

[1] University of Stuttgart, IPVS, 70569 Stuttgart, Germany, {firstname.lastname}@ipvs.uni-stuttgart.de
[2] University of Stuttgart, GSaME, 70569 Stuttgart, Germany, peter.reimann@gsame.uni-stuttgart.de

doi:10.18420/BTW2023-16

in real-world data in the sense that classification patterns may vary among the groups and that the groups are imbalanced in the data [HRM19, Me21, SG21]. This leads to very different analysis results when performing the same analysis on the entire data or on each group separately [Me21, SG21]. Both characteristics together are present in several use cases, e. g., in use cases across the industrial value chain [KBT11, Su14, Wu16, KM16, HRM19, Gr22] or in medical use cases [KU15, SGG18, Ch20, MGTM20, SG21].

However, it is typically not possible to obtain representative benchmark data that contain both a multi-class imbalance and heterogeneous groups. Real-world datasets are usually not publicly available, e. g., because they constitute sensitive patient information or due to privacy concerns. Obtaining data from publicly available repositories, e. g., OpenML [Va14], is also not a feasible option, since they do not reflect both characteristics to the same extent as they occur in data of real-world use cases. To make meaningful evaluations and benchmarks of classification algorithms, it should furthermore be possible to control the manifestations of both characteristics in the data, i. e., the degree of class imbalance or the heterogeneity and imbalance of domain-specific groups. This is however neither possible with individual real-world datasets nor with data from common repositories. Therefore, a more rigorous approach is to generate synthetic data with both multi-class imbalance and heterogeneous groups. Existing data generators (e. g., [SH05, Fr11, FS18, Ig19, Gu03]) are able to generate data characteristics that are based on statistical properties. That is, they can generate data with different degrees of multi-class imbalance. Yet, to generate data with heterogeneous groups, domain knowledge about the groups from real-world application domains is required. However, existing data generators do not use such domain knowledge and thus are not able to generate data with heterogeneous groups. In addition, when the data to be generated has to contain both a multi-class imbalance and heterogeneous groups, a data generator has to ensure that all dependencies between both data characteristics are properly reflected within the data. For instance, the class imbalance not only has to be reflected within the whole dataset, but also within each subset of the respective groups.

In this paper, we propose an approach to generate data synthetically that mitigates the drawbacks of existing data generators and publicly available data repositories. That is, our approach is capable of generating numerical and categorical data with both multi-class imbalance and heterogeneous groups. An important design guideline of our approach is the applicability in many different domains. Our contributions include the following:

- Our approach is the first that is capable of generating numerical and categorical data with domain-specific heterogeneous groups. To realize this, we use a taxonomy that organizes such groups in a hierarchical structure. A taxonomy is the simplest form of knowledge models and can thus be found in a wide range of domains. In consequence, our approach can be used in a variety of domains as well.
- We use probability distributions to reflect the imbalances of real-world groups and classes. To this end, we state requirements that such probability distributions have to fulfill. In our approach, we use the Zipf distribution as it fulfills all requirements.

- We propose a two-step procedure to generate the data synthetically based on the taxonomy and the probability distribution. First, we traverse the taxonomy top-down to specify important data characteristics regarding the group distribution. Second, we specify the class distributions among the groups and generate the data bottom-up.
- In our evaluation, we show that the data our approach may generate comprise both data characteristics. To this end, we assess the characteristics with different metrics such as classification complexity, statistics, or a detailed view on classification accuracy. In addition, we show to which extent individual parameters of our approach influence the characteristics and the aforementioned metrics.

The rest of this paper is structured as follows: In Section 2, we define the characteristics that we generate with our proposed approach. We examine limitations of existing data repositories and data generators in Section 3. In Section 4, we describe our method to generate data synthetically and how this method may be parameterized to incorporate different degrees of each characteristic. We discuss the results of evaluating our data generation approach with different parameter configurations in Section 5. Section 6 finally concludes our work.

## 2 Data Characteristics

We assume a classification problem with $c > 2$ class labels, i. e., a multi-class problem with classes $C = \{C_1, ..., C_c\}$. This problem consists of a dataset $\mathcal{X} = \{(x_i, y_i)\}_{i=1}^{n}$ with $n$ tuples, where each tuple contains an instance $x_i$ of the data and a class label $y_i \in C$. Here, $x_i$ is a feature-vector from a $f$-dimensional feature space $\mathcal{F} = \{F_1, .., F_f\}$. Each feature $F_i$ either has categorical or numerical values. The problem is then to generate data that comprise the data characteristics multi-class imbalance (DC1) and heterogeneous groups (DC2).

### 2.1 Multi-Class Imbalance (DC1)

A prevalent challenge in literature and in many practical scenarios is *multi-class imbalance* [HG09, WY12, Wu16]. Here, certain classes are represented more often in $\mathcal{X}$ than other classes. The classes occurring more often $C^+ \subset C$ are called majority classes, while the less frequent classes $C^- = C \setminus C^+$ are called minority classes. Machine learning algorithms aim to optimize the overall accuracy of the predictions for the whole dataset $\mathcal{X}$. Majority classes have a big share of all instances in $\mathcal{X}$ so that the overall accuracy highly correlates with the accuracy of predictions for these majority classes. Thus, machine learning algorithms tend to ignore the instances $\mathcal{X}^-$ of minority classes and are therefore biased towards majority classes. However, making accurate predictions for minority classes is in many real-world use cases even more important [Wu16]. In data-driven medical diagnoses, they may represent rare, but dangerous or even lethal diseases that must be detected with the highest accuracy possible [Ch20]. While literature comprises many approaches that may handle binary class imbalance, e. g., approaches to over- or under-sampling, these approaches are not able to deal with multi-class imbalance [WY12, Fe13]. Further, multi-class imbalance often comprises accompanying symptoms that likewise lead to challenges for classification algorithms [HG09, WY12]. For instance, this concerns overlapping classes, i. e., instances from classes that are next to each other (a.k.a border points [HB02]).

## 2.2 Heterogeneous Groups (DC2)

Data in real-world scenarios often represent the observations for diverse groups of entities. In industrial use cases, these groups of entities constitute the various product groups, e. g., a vehicle engine can be differentiated by 'Diesel' or 'Gasoline' engines, which each may be further divided into four- and six-cylinder engines [HRM20]. In medical applications, different groups of patient populations exist, e. g., patients with different gender types [ULP19, WLL21] or with different skin colors [Ch20]. Hence, data from real-world use cases typically comprise $k$ domain-specific groups $G_1, ..., G_k \subset \mathcal{X}$ such that $\bigcup_{i=1}^{k} G_i = \mathcal{X}$ and $G_i \cap G_j = \emptyset$ for $i \neq j$. Further, each group $G_i$ can comprise multiple classes $C_{i1}, ..., C_{ic}$, while each class may be included in multiple groups. These real-world groups and their data are heterogeneous in the sense that they are distributed in an imbalanced way in the dataset $\mathcal{X}$ (DC2a) and show heterogeneous class patterns (DC2b).

**Imbalanced Groups (DC2a):** Data from real-world use cases often comprise imbalanced groups, i. e., some groups occur more frequently than others. In medical applications, different groups of patients are typically more frequent than others, e. g., patients with lighter skin colors are typically more frequent than patients with darker skin colors [Ch20]. For industrial use cases, certain product groups occur more often than others [Su14, Wu16, KM16, HRM20], e. g., 'Gasoline' engines are more frequent than 'Diesel' engines. Thus, the dataset $\mathcal{X}$ comprises majority groups $G^+ \subset \mathcal{X}$ that appear more frequently than minority groups $G^- = \mathcal{X} \setminus G^+$. Learning algorithms tend to favor the majority groups $G^+$ as these comprise much more instances of the data. Further, the data may comprise underrepresented minority groups $G^-$ that occur very rarely and may thus be ignored by classification algorithms. This is also known as representation bias in literature [Me21, SG21].

**Heterogeneous Class Patterns (DC2b):** Real-world use cases often exhibit a heterogeneity of class patterns, i. e., a single class is described by different patterns in the data subsets of different groups $G_i$. An example in medical applications is that the symptoms for specific types of skin cancer vary for different skin colors [Ch20], i. e., the different groups (skin colors) exhibit different patterns (symptoms) for the same class (cancer type). In many industrial use cases, the patterns for the same class likewise vary across different product groups [Su14, KM16, Wu16, HRM20, Wi20]. In different groups, the same class may for instance have different value ranges for the same feature [HRM20]. This heterogeneity of class patterns usually leads to an aggregation bias [Me21, SG21]. That is, a particular data analysis carried out on the entire dataset $\mathcal{X}$ yields different results than the very same analysis performed on each group $G_i$ separately.

Tab. 1: Overview of related repositories and data generators regarding their data characteristics and domain-independence. A '✓' means that the characteristic can be generated or that the criterion is fulfilled, while a '✗' means the opposite.

| Category | Examples | Domain-Independent | DC1: Multi-class Imbalance | DC2: Heterogeneous Groups |
|---|---|---|---|---|
| Repositories | UCI [DG17], OpenML [Va14], KEEL [Al11], Kaggle, etc. | ✗ | ✓ | ✗ |
| Domain-specific Data Generators | fraud detection [LKJ02], health care [DC19], production-oriented [Fe20], etc. | ✗ | ✓ | ✗ |
| Data Augmentation | GANs [GBC16, RHW21], SDV [PWV16], etc. | ✗ | ✓ | ✗ |
| Domain-agnostic Data Generators | Clustering [SH05, Fr11, FS18, Ig19], Classification [Gu03], Scikit-learn | ✓ | ✓ | ✗ |

## 3 Related Work

Table 1 summarizes our key findings of related work, which we discuss in the following.

**Machine Learning Repositories:** Literature often makes use of data from publicly available machine learning repositories to develop and evaluate novel machine learning algorithms. Several machine learning repositories exist, e. g., OpenML [Va14], KEEL [Al11], Kaggle[3], and the UCI ML Repository [DG17]. These repositories include around 3500 datasets. The above mentioned repositories together only offer 40 data sets with more than 10 classes and with at least a moderate multi-class imbalance (DC1). Regarding heterogeneous groups (DC2), some works in the area of fair machine learning consider the adult[4] or COMPASS[5] dataset, where different groups of gender or skin color are present, e. g., 'male' and 'female' or 'white' and 'black' [ULP19, WLL21]. Yet, these datasets typically contain only two or up to four groups [ULP19, WLL21]. For industrial use cases, there may be thirty [HRM20] or even thousands [Su14] of different groups in the data. Hence, data from publicly available repositories do not have heterogeneous class patterns or imbalanced groups (DC2) to the same extent as found in real-world use cases [Su14, HRM20, Ch20]. Further, each dataset is specific for a certain domain and thus we evaluate the repositories as domain-dependent.

**Domain-specific Data Generators:** As common repositories do not offer data containing both data characteristics, the next possibility is to generate data synthetically. Literature comprises different synthetic data generators that focus on specific domains, e. g., fraud detection [LKJ02], health care applications [DC19], or production-oriented data [Fe20]. These works are typically structured into two steps: First, they define a specific data model for the domain at hand and second, generate the data according to this data model. Yet,

[3] Kaggle datasets: `https://www.kaggle.com/datasets`

[4] Adult dataset: `https://archive.ics.uci.edu/ml/datasets/adult`

[5] Machine Bias article: `https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing`

these data models are specific to the given domain. For instance, Fernandes et al. pre-define 15 production-oriented features for the data model such as customer demand, total parts, or available time [Fe20]. Hence, these generators are only applicable to the domains for which appropriate data models are available [SB21]. Further, none of them focus on DC2. Although they use domain knowledge in terms of the data model, this data model usually does not consider the domain-specific groups or their heterogeneity. Therefore, they are for instance not able to generate different class patterns for these groups (see Table 1).

**Data Augmentation from Real-World Data:** The next group of synthetic data generators requires existing real-world data and uses this data as basis for data augmentation [RHW21, PWV16]. These approaches take a sample of the existing data, learn the distribution of the data sample and then generate new data from the learned distributions [RHW21,PWV16]. They typically employ generative adversarial networks (GANs) [GBC16], which are based on two networks: A generative network learns to map a latent space to a data distribution, and a discriminative network then draws samples from the learned distribution. However, these approaches require existing real-world data to model the distribution of DC1 and DC2. Since available data only covers DC1 (see Table 1), approaches to data augmentation can only generate new data that resembles an existing multi-class imbalance. Yet, they are not able to generate heterogeneous groups (DC2).

**Domain-Agnostic Data Generators:** The last group of related work covers domain-agnostic data generators [SB21], i. e., generators that are independent of specific domains and that do not require real-world data as basis. Literature comprises domain-agnostic generators for clustering [SH05,Fr11,FS18,Ig19] or for classification tasks [Gu03]. The approaches can vary the number of instances $n$, the number of features $f$, and the number of classes $c$. Most of them are able to generate data with multi-class imbalance (DC1). For example, scikit-learn[6] offers a data generator for classification that is taken from Guyon [Gu03]. To generate multi-class imbalance, users can pass in weights $w = \{w_1, ..., w_c\}$ for each class. These weights define the share of instances for each class, i. e., $w_i \in [0; 1]$ and $\sum_{i=1}^{c} w_i = 1$. The approach then generates the instances of each class separately. To this end, it generates for each class one or several clusters, where each class $c_i$ has $n \cdot w_i$ instances. It can generate multiple clusters for each class, but each cluster has the same number of instances. Hence, the domain-agnostic generators do not generate clusters (or groups) in the data with varying cluster sizes, i. e., imbalanced groups (DC2a). Moreover, they do not generate a heterogeneity of the class patterns among different clusters (DC2b). The reason is that they do not use domain knowledge about the groups and their distributions.

Summarizing related work, none of the related approaches is able to generate datasets that show both data characteristics (see Table 1), i. e., multi-class imbalance (DC1) and heterogeneous groups (DC2). Nevertheless, our approach employs existing domain-agnostic data generators to generate data with DC1 and extends them to also generate data with DC2, as well as with different manifestations of both DC1 and DC2.

[6] Scikit-learn data generator: `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification`

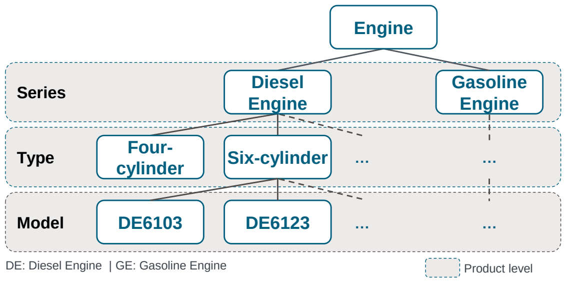


Fig. 1: Simplified excerpt of a taxonomy that defines hierarchical relationships of various engine types.

## 4 Approach to Synthetic Data Generation of DC1 and DC2

In this section, we describe our approach to generate data synthetically that comply with both data characteristics DC1 and DC2. Our approach can vary the number of instances $n$, features $f$, and classes $c$. Furthermore, our approach has the following parameters that influence the manifestations of DC1 and DC2:

- $s_C \in \mathbb{R}_{\geq 0}$: Parameter to control the extent of imbalance for the classes in the generated data (DC1).
- $s_G \in \mathbb{R}_{\geq 0}$: Parameter to control the extent of imbalance for the groups (DC2a).
- $co \in \mathbb{R}_{\geq 1}$: Controls the class overlap between the groups. For $co = 1$, the classes are distributed disjoint across the groups, i. e., each class occurs only in one group. The higher the value, the more classes are in one group und thus finally also more classes tend to overlap across the groups. Then, the classification task gets more difficult.
- $gs \in \mathbb{R}_{\geq 0}$: Parameter to control the group separation. It mainly influences the heterogeneity of the groups (DC2b). A low value means that the groups highly overlap with respect to the feature ranges of their instances. Higher values indicate more clearly separable groups.
- $cf \in \{1, 2, ..., f\}$: Number of characteristic features for each group, i. e., the number of features for which we separate the groups with $gs$ (DC2b).

In the following sections, we first describe the domain knowledge model that we use to generate data that comprise real-world groups (DC2b). Subsequently, we examine probability distributions to generate data with imbalanced distributions for the classes (DC1) and for groups (DC2a). Finally, we detail on our algorithms to generate the data synthetically.

### 4.1 Taxonomy

For the generation of data with the characteristic DC2, we use information and characteristics regarding groups occurring in an application domain. Usual ways to model domain-specific groups and their relationships are knowledge graphs [Ho21] and semantic nets, e. g., taxonomies, thesauruses, or ontologies [So91]. Such models commonly organize the entities of a domain, amongst others, via hierarchical relationships of superordinate and subordinate groups. Subordinate groups, i. e., child nodes in the hierarchy, are more specific than their associated superordinate groups, i. e., the parent nodes [So91]. Hence, data related to any subordinate child group is a subset of the data of its parent group.

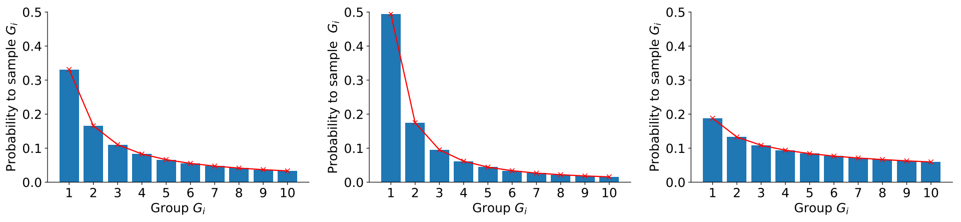
Our approach to generate data solely builds on hierarchical relationships among domain-specific groups. We do not need other more complex types of relationships as found in thesauruses, ontologies, or knowledge graphs. Hence, it is sufficient to use taxonomies, which already come with hierarchical relationships. As taxonomies are the simplest form of a semantic net, the effort to create them is moderate. Hence, they are pre-defined in various domains, e. g., for product families [AK04, Su14, HRM20] or skin colors of patients [Ja04]. Even if not present, literature comprises several ontology learning approaches to extract hierarchies from data, e. g., hierarchical clustering or association rule discovery [Ci09].

We define the hierarchical tree structure of a taxonomy as $T = (V, E)$ with nodes $V = \{v_1, ..., v_t\}$ and edges $E \subseteq V \times V$. $T$ is a directed, acyclic tree with exactly one root node and where each child node has exactly one parent node. An edge $E_{ij} = (v_i, v_j)$ furthermore represents the hierarchical relationship of nodes $v_i$ and $v_j$, i. e., $v_i$ is the parent node or the superordinate concept of $v_j$. This taxonomy model is also often encoded in the data itself. Therefore, the levels of $T$ are encoded by level-specific key features. A node $v_i$ on a level $l$ has a distinct value in the key feature specific to level $l$.

**Example:** Figure 1 shows an example of a taxonomy that defines the hierarchical relationships of different engine types of motor vehicles. This taxonomy is adapted from a recent work regarding the end-of-line testing of complex truck engines [HRM20]. In this example, an engine is first distinguished between Diesel or Gasoline engines, and further divided by its engine type and model. Thus, we have three level-specific key features *series*, *type*, and *model*. The level-specific feature *series* may have one of the distinct values 'Diesel Engine' or 'Gasoline Engine'. Furthermore, every model 'DE6123' has also the type 'Six-cylinder', and is of series 'Diesel Engine', i. e., the edges describe the hierarchical relationships.

### 4.2 Probability Distribution

In our data generation, we use probability distributions ($PD$) to reflect the imbalances of the classes (DC1) and groups (DC2a). That is, we use $PD$ in two ways: First, to assign the number of instances for each class, and second, to assign the number of instances and classes among the groups. Therefore, the probability distribution has to (i) sample *univariate*



(a) $s = 1$. (b) $s = 1.5$ (c) $s = 0.5$

Fig. 2: Probability distribution to sample a group from 10 possible groups with the Zipf distribution. We vary the parameter $s$ with (a) default value $s = 1$, (b) a more imbalanced setting with $s = 1.5$, and (c) a more balanced setting with $s = 0.5$.

random variables, (ii) reflect *imbalanced distributions* of the groups or classes, (iii) be *discrete*, as we sample integer values, i. e., the number of instances for the groups or classes, (iv) allow for *specifying the number of possible values* to sample from, e. g., we want to sample from 10 groups or 100 classes, and (v) allow to *parameterize* the distribution such that we are able to generate different manifestations of DC1 and DC2a.

A distribution that fulfills these criteria is the Zipf distribution from the family of exponential or power-law distributions [Sc12]. We examined several distributions that fulfill our criteria, e. g., Boltzman and Poisson [Sc12], but Zipf obtained the most similar data distribution compared to real-world data [HRM19, HRM20]. The first input is the number of classes $c$ to generate. Furthermore, we have information about the groups from the taxonomy, i. e., we assume we have $k$ groups $G_1, G_2, ..., G_k$ that are the leaf nodes of the taxonomy. We also assume that a ranking exists that orders the groups by the number of samples $|G_i|$ to generate for each group $G_i$, i. e., $|G_1| \geq |G_2| \geq ... \geq |G_k|$. Hence, given the number of groups $k$, the rank $r_i \in \{1, 2, ..., k\}$ of a group $G_i$, and the exponent $s \in \mathbb{R}_{\geq 0}$ that parameterizes the Zipf distribution, the probability to sample an instance for group $G_i$ can be expressed as

$$P_{k;s}(r_i) = \frac{1}{r_i^s H_{k,s}}, \quad (1)$$

where $H_{k,s} = \sum_{j=1}^{k} \frac{1}{j^s}$ is the $k$-th harmonic number [Sc12]. Thereby, we can control the imbalance degree of the classes or groups using the parameter $s$ of the distribution. As we use $PD$ for two different purposes in our approach, we use the parameters $s_C$ and $s_G$ to describe the imbalance degrees for the classes and groups, respectively.

**Example:** Figure 2 shows an example of the Zipf distribution and the influence of the parameter $s$. We focus on the distribution of different groups, but it is analogous for the distribution of the classes. In this example, we assign the number of instances to $k = 10$ groups. We show the probability to sample an instance for the $i$-th group on the y-axis. Figure 2a shows the probability to sample an instance for a group with the default setting $s = 1$. The probability to sample the first group is around 35% and for the second group

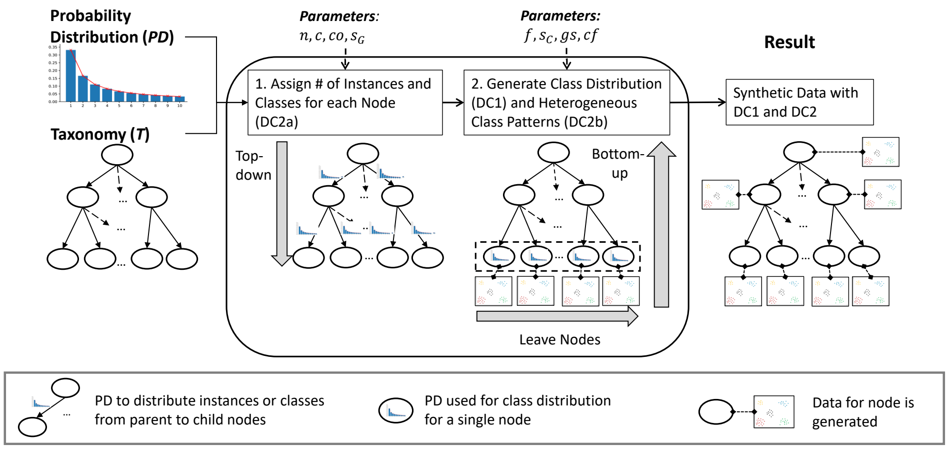

Probability Distribution (*PD*)
Taxonomy (*T*)
*Parameters:* $n, c, co, s_G$
*Parameters:* $f, s_C, gs, cf$
1. Assign # of Instances and Classes for each Node (DC2a)
2. Generate Class Distribution (DC1) and Heterogeneous Class Patterns (DC2b)
Top-down
Bottom-up
Leave Nodes
Result
Synthetic Data with DC1 and DC2
PD to distribute instances or classes from parent to child nodes
PD used for class distribution for a single node
Data for node is generated

Fig. 3: General overview of the two main steps of our approach.

around 17.5%. Hence, if we sample 100 times on this distribution, we have one group that gets around 35 instances, a second group with around 18 instances, and so on. In particular, for $s = 1$, the element that occurs most often occurs twice as often as the second-most occurring element [Sc12]. Furthermore, we show the effect of varying the exponent $s$: Figure 2b shows a more imbalanced setting with $s = 1.5$ and Figure 2c a more balanced setting with $s = 0.5$. In general, a setting with $s < 1$ leads to more balanced and $s > 1$ to more imbalanced distributions. A value $s = 0$ describes an equal probability for each group.

## 4.3 Data Generation

Figure 3 gives a general overview of our two-step approach: First, we generate imbalanced groups (DC2a). To this end, we assign the number of instances and classes for each group, i. e., for each node in the taxonomy. Second, we generate an imbalanced class distribution for each node (DC1) and ensure the heterogeneity of the class patterns among the groups (DC2b). Note that due to DC2b, the heterogeneous class patterns are specific for each group. Thus, we first have to generate the groups and subsequently the classes and patterns within them so that we can directly ensure that the class patterns are different between the groups.

### 4.3.1 Assign Number of Instances and Classes

In the first step, we aim to generate imbalanced groups (DC2a), i. e., we assign the number of instances and classes to all groups of the taxonomy. Algorithm 1 outlines the procedure for this first step. The inputs are the number of instances ($n$), number of classes ($c$), the class overlap across the groups ($co$), the taxonomy ($T$), the probability distribution ($PD$), and the parameter for the group imbalance degree ($s_G$). First, we initialize the root node in line 1. Since the root node should comprise the entire dataset $\mathcal{X}$, we assign the overall numbers of instances $n$ and classes $c$ to the root node. Furthermore, we initialize the set of $nodes$ that we traverse in the following with the root node (line 2). Hence, as long as we have nodes to traverse (line 3), we pick and remove the next $node$ from the nodes set (line 4). If $node$ does not have any child nodes, i. e., $n\_children = 0$, we continue with the next iteration of the while loop (lines 6 - 8).

If the current $node$ has children, we assign the number of instances and the number of classes for all its child nodes. To this end, we use the $PD$ by calling the function $\textsc{Sample\_PD}(PD, node.n, n\_children, s_G)$ (line 9). We use this function to distribute the number of instances of a parent node ($node.n$) among its child nodes (lines 16 - 23). For each individual instance, we decide to which group it belongs using $PD$. Thus, we sample $node.n$ times from the $PD$ and sample each time one of the child nodes $1, ..., n\_children$. We control the imbalance of sampling the groups with the parameter $s_G$. In the $\textsc{Sample\_PD}()$ function, we first initialize a $counter$ list that keeps track of the number of instances for each group (line 17). Subsequently, we sample one of the groups from $PD$ and update the count for that group (lines 19 and 21). Finally, we return a list $n\_instances$ that contains for each group the number of instances according to the $PD$. As this is an exponential distribution, we get an imbalanced distribution of the number of instances across all nodes for $s_G > 0$.

In line 10, we do the same procedure for the classes, i. e., we derive a list $n\_classes$ that specifies the number of classes for each group. However, as mentioned in Section 2.2, classes typically occur in multiple groups. Thus, we add a factor of $co \geq 1$ to control the number of classes among the groups. For $co = 1$, the classes are disjoint among the groups. For $co > 1$, classes may occur in multiple groups. In line 11, we set the number of instances and the number of classes as attributes of the child nodes. That is, we update the attribute of the child nodes with the corresponding samples of the lists $n\_instances$ and $n\_classes$ in the $\textsc{Update}(children, n\_instances, n\_classes)$ function (lines 24 - 30). In line 12, we add the child nodes to the $nodes$ set to traverse them as well. Finally, we return the modified taxonomy, where each node contains the number of instances and the number of classes.

### 4.3.2 Assign Class Distribution and Generate Data

In the second step, we assign the class distributions and generate the data in a bottom-up manner. To this end, we use the information that we assigned in the previous step to each node. We first generate the data on all leaf nodes and subsequently pass the data upwards to the parent nodes.

Algorithm 2 outlines our procedure. First, we retrieve the leaf nodes from the taxonomy (line 1). Further, we initialize a key-value map to store the current feature limits (line 2). Then, we iterate over each leaf node (lines 3 - 17). We first retrieve the number of instances, number of classes, and the actual class labels from each leaf node (line 4). In line 5, we

**Algorithm 1** Algorithm to assign the number of instances and classes.

```
Input: T: Tree-structured taxonomy,
    $n$: Number of instances for the entire data,
    $c$: Number of classes for the entire data,
    $co$: Class overlap across groups,
    $PD$: Probability distribution,
    $s_G$: Value for the group imbalance degree used by $PD$.
Output: T: Tree-structured taxonomy that has the number of instances and classes assigned as attributes on each
    node.

    ▷ Initialize root node and nodes set
 1: $root.n \leftarrow n$; $root.c \leftarrow c$; $root.classes \leftarrow \{1, ..., c\}$;
 2: $nodes \leftarrow \{root\}$
    ▷ Iterate while we have nodes
 3: while $nodes \neq \{\}$ do
        ▷ Get and remove node from nodes set
 4:     $node \leftarrow nodes.pop()$
 5:     $n\_children \leftarrow |node.children|$
 6:     if $n\_children == 0$ then
 7:         continue                                        ▷ Leaf node, so continue with next node
 8:     end if
        ▷ Draw the number of instances for each child node
 9:     $n\_instances \leftarrow$ SAMPLE_PD($PD, node.n, n\_children, s_G$)
        ▷ Draw the number of classes for each child node
10:     $n\_classes \leftarrow$SAMPLE_PD($PD$, $node.c * co$, $n\_children$, $s_G$)
        ▷ Update $n\_instances$ and $n\_classes$ of child nodes
11:     UPDATE($node.children, n\_instances, n\_classes$)
12:     $nodes$.APPEND($node.children$)
13: end while
14: return T
15: procedure SAMPLE_PD($PD$, $k$, $n\_groups$, $s$)
16:     $counter \leftarrow [0, ..., 0]$                    ▷ Initialize counter of size $n\_groups$
17:     for $i = 1, ..., k$ do
18:         $group \leftarrow \mathrm{PD}(n\_groups, s, i)$              ▷ Sample group from $PD$
19:         $counter[group] \leftarrow counter[group] + 1$
20:     end for
21:     return $counter$
22: end procedure
23: procedure UPDATE(children, n_instances, n_classes)
24:     for $i = 1, ..., |children|$ do
25:         $child \leftarrow children[i]$
26:         $child.n \leftarrow n\_instances[i]$
27:         $child.c \leftarrow n\_classes[i]$
28:     end for
29: end procedure
```

ensure the multi-class imbalance (DC1). Here, we assign the class occurrences, i. e., how often a particular class occurs in the dataset. To this end, we use $PD$ to decide for each instance the associated class, similar as for the groups in Algorithm 1. Hence, we sample $n$ times one of the classes $1, ..., c$ via the distribution $PD$ and the function Sample_PD (cf. lines 18 - 25 in Algorithm 1). In lines 6 and 7, we ensure the heterogeneity of the class patterns (DC2b). First, we pick $cf$ characteristic features from the current group, i. e., the features that separate the current group from all other groups. To separate the groups, we ensure that the current group has different value ranges for the characteristic features than the other groups. To this end, we increment the feature limits with the $gs$ parameter to control the differences in the value ranges between the groups.

In line 8, we generate the data using the input parameters $n$, $c$, the number of features $f$, and the list $class\_occurrences$. That is, we generate the feature values for all instances with their associated class labels. To this end, we can use any multivariate probability distribution to generate the data that supports these inputs as parameters. As existing domain-agnostic data generators (cf. Section 3) already support generating data using different probability distributions, we can use them for that purpose. For example, the approach from Guyon [Gu03] may be used to generate the feature values and feature ranges for specific class labels according to a Gaussian distribution. However, this multivariate probability distribution can also be altered by the user to capture different feature correlations of real-world applications.

To guarantee the heterogeneity of class patterns, we ensure that each class has different value ranges in different groups for certain characteristic features. Such feature dependencies in form of characteristic features also appear frequently in industrial application scenarios that comprise heterogeneous groups [Wu16, HRM19, KRM19]. Therefore, we ensure in lines 9 and 10 that the groups have different value ranges according to the picked features for each group and the current $feature\_limits$. To this end, we normalize the feature values of all instances, i. e., we normalize the values of each feature in $\mathcal{X}$ into [0;1] (line 9). Subsequently, we add the current limits of the features (line 10). In line 11, we store $\mathcal{X}$ in the current node. In lines 12 - 15, we update the data and the class labels of the parent nodes. That means we traverse the parent node and append the data $\mathcal{X}$ of the child node to the currently stored data of the parent node. Finally, we return the taxonomy, where we set the data for each node.

## 5 Evaluation

In this section, we discuss the evaluation results for our approach, i. e., whether our approach is able to generate data synthetically that comply with the data characteristics DC1 and DC2. First, we describe the setup of our evaluation. Subsequently, we discuss to which extent our data generator can generate different manifestations of the data characteristics DC1, DC2a, and DC2b using different parameterizations.

**Algorithm 2** Algorithm to generate the data bottom-up.

**Input:** $f$: Number of features to generate,
$s_C$: Imbalance degree for the class distribution,
$gs$: Group separation,
$cf$: Number of characteristic features to use for each group,
$PD$: Probability distribution,
T: Tree-structured taxonomy, where the numbers of instances and classes as well as the class occurrences are defined for each node with Algorithm 1.

**Output:** T: Tree-structured taxonomy, where we assigned for each leaf node how often each class occurs.

```
1: $nodes \leftarrow$ GET_LEAF_NODES$(T)$
2: $feature\_limits \leftarrow$ INIT_DICTIONARY$(\{1, ..., f\}, 0, 1)$
3: for $node \in nodes$ do
4:     $n \leftarrow node.n$; $c \leftarrow node.c$; $classes \leftarrow node.classes$;
       ▷ Draw the number of instances for each class
5:     $class\_occurrences \leftarrow$ SAMPLE_PD$(PD, n, classes, s_C)$
       ▷ Pick characteristic features for this group
6:     $features \leftarrow$ PICK_RANDOM_FEATURES$(\{1, ..., f\}, cf)$
       ▷ Separate the groups
7:     INCREMENT$(feature\_limits, features, gs)$
       ▷ Actual data generation for each leaf node
8:     $\mathcal{X} \leftarrow$ GENERATE_DATA$(n, c, f, class\_occurrences)$
9:     $\mathcal{X} \leftarrow$ Normalize $X$ into $[0; 1]$
       ▷ Add for each feature its current limits
10:    $\mathcal{X} \leftarrow \mathcal{X} + feature\_limits$
       ▷ Store data in current node
11:    $node.X \leftarrow \mathcal{X}$;
       ▷ add $\mathcal{X}$ to parent node
12:    while HAS_PARENT$(T, node)$ do
13:        $node \leftarrow$ GET_PARENT$(T, node)$
14:        $node.X \leftarrow$ APPEND$(node.X, \mathcal{X})$
15:    end while
16: end for
17: return T
```

## 5.1 Evaluation Setup

**Implementation.** Our prototypical implementation is available on Github[7]. For the evaluation, we use a taxonomy that we derived from a real-world use case regarding end-of-line testing of complex truck engines [HRM20]. A simplified excerpt of this taxonomy is shown in Figure 1. The taxonomy has three levels and 26 groups at the bottom level of the hierarchy. For more details on the taxonomy, we refer to our repository.

**Evaluation of the data characteristics.** The goal of our evaluation is to show that the generated data of our approach comply with the data characteristics DC1, DC2a, and DC2b. We evaluate the presence of each data characteristic with different evaluation measures, i. e., class imbalance measures for DC1, group imbalance measures for DC2a, and complexity

[7] Prototypical implementation: `https://github.com/IPVS-AS/DataGenerator`

Tab. 2: Overview of parameters that we discuss regarding their influence on the data characteristics. The bold values indicate the parameters that we vary for the respective characteristics, while we use default values for the other parameters.

| Data Characteristics | Parameter values | | | |
|---|---|---|---|---|
| | Class imbalance ($s_C$) | Group imbalance ($s_G$) | #charact. features ($cf$) | Group separation ($gs$) |
| Multi-Class Imbalance (DC1) | **{0, 1,2†, 3, 4, 5}** | 1 | 10 | 0.25 |
| Group Imbalance (DC2a) | 2 | **{0, 0.5, 1, 1.5†, 2}** | 10 | 0.25 |
| Heterogeneity of Class Patterns (DC2b) | 2 | 1 | **{1, 5, 10†, 15, 20, 25, 30}** | **{0†, 0.05, 0.1, 0.25, 0.5, 0.75, 1}** |

† Parameter values that lead to similar statistics as the real-world data in the work of Hirsch et al. [HRM19, HRM20].

Tab. 3: Gini coefficient values for the generated datasets with varying $s_C$ parameter.

| Class Imbalance ($s_C$) | $s_C = 0$ | $s_C = 1$ | $s_C = 2$ | $s_C = 3$ | $s_C = 4$ | $s_C = 5$ |
|---|---|---|---|---|---|---|
| Gini Coefficient | 28% | 41% | 60% | 69% | 72% | 73% |

measures for DC2b. To this end, we also vary the parameter values of our data generator to show that it is capable of generating different manifestations of the data characteristics. This is an essential requirement for generating data that may serve as basis for benchmarks of classification algorithms. Furthermore, we also compare the statistics of the generated data with the statistics of a real-world data set of Hirsch et al. [HRM19, HRM20].

**Parameters.** Our data generator has eight parameters in total. As our goal is to evaluate which manifestations of the data characteristics it can generate, we only vary the parameters that have a strong influence on these characteristics. Therefore, we generate data with fixed parameters $n = 1000$ instances, $f = 40$ features, $c = 30$ classes, and $co = 1.5$. Table 2 shows the parameters that we vary to study the influence on individual data characteristics. To evaluate the manifestation of a single data characteristic, we only vary and discuss the results for the parameters that influence this particular characteristic, using default values for the other parameters (cf. Table 2).

## 5.2 Evaluation of Multi-Class Imbalance (DC1)

We evaluate the presence of the data characteristic DC1 w.r.t. the class imbalance and the accuracy for minority and majority classes of a classification model.

**Class imbalance.** There is no consensus on proper statistical metrics to determine the degree of class imbalance within data [Fe13]. Yet, an often-used metric for inequality that takes the value of 100% in case of total imbalance and 0% for total balance is the Gini coefficient [Co00]. Thus, we use the Gini coefficient to measure the imbalance of the classes for the generated datasets.

Table 3 shows the Gini coefficients for the generated datasets with different $s_C$ parameter values. We observe that the Gini coefficient values for the generated data vary from 28% for

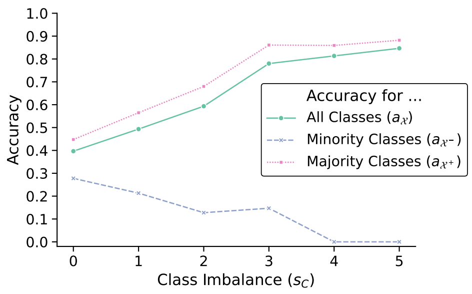


Fig. 4: Overview of accuracy for all classes ($a_\mathcal{X}$), and separately for minority ($a_{\mathcal{X}^-}$) and majority classes ($a_{\mathcal{X}^+}$) for the different $s_C$ values.

$s_C$ = 0 to 73% for $s_C$ = 5. Thus, the class imbalance is increasing with an increasing value of the $s_C$ parameter. We also observe that the Gini coefficient is increasing faster for lower $s_C$ values, i. e., it is increasing by 19%-points from $s_C$ = 1 to $s_C$ = 2, but only by 2%-points from $s_C$ = 4 to $s_C$ = 5. The reason why the coefficient does not increase as much for higher $s_C$ values is due to the calculation of the Gini coefficient. For example, to achieve a Gini coefficient of 100%, the generated data would contain instances for solely one class [Co00]. So, for the generated data to have a higher Gini coefficient, some of the classes must not appear in the data at all. However, in our implementation, we ensure that each class occurs at least once. The real-world data in the work of Hirsch et al. [HRM19, HRM20] shows a Gini coefficient of 55%. Thus, with $s_C$ = 2, we can generate data with a similar class imbalance as real-world data that also comprise DC1 and DC2.

**Accuracy of minority and majority classes.** As described in Section 2.1, the accuracy of a classifier typically correlates with the accuracy for the majority classes in multi-class imbalance problems. In particular, the accuracy for minority classes usually decreases as the degree of class imbalance increases [HG09, WY12]. Therefore, we examine whether the generated data by our generator also exhibit this trend. To measure the accuracy, we split each generated dataset into 70% training data $\mathcal{X}_{train}$ and 30% test data $\mathcal{X}_{test}$. Thereby, we preserve the same class distribution in both data subsets. Subsequently, we train a classifier $M_\mathcal{X}$ on $\mathcal{X}_{train}$ and denote the accuracy of $M_\mathcal{X}$ on $\mathcal{X}_{test}$ as $a_\mathcal{X}$. We also denote with $a_{\mathcal{X}^+}$ the accuracy among only the instances of the majority classes and with $a_{\mathcal{X}^-}$ for the minority classes. We declare a class as minority class if it has less instances than the median number of instances of all classes and otherwise as majority class. We use Random Forest as classification model due to its robustness regarding the characteristics [HRM19, HRM20].

Figure 4 shows the accuracy of the minority and the majority classes for each generated dataset. We observe that the more imbalanced the data, the higher is the accuracy for the whole dataset ($a_\mathcal{X}$) and for the majority classes ($a_{\mathcal{X}^+}$). The reason for this high correlation between $a_\mathcal{X}$ and $a_{\mathcal{X}^+}$ is that the majority classes have by far the biggest share of instances of the data. So, they also contribute much more to the overall accuracy than the minority

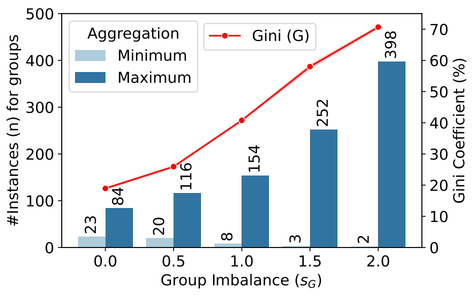


Fig. 5: Overview of the number of instances per group as well as the Gini coefficient values for different $s_G$ values.

classes. However, the accuracy of the minority classes decreases with an increasing class imbalance. As mentioned above, this is the expected behavior for the accuracy in multi-class imbalance problems. Hence, when we control the imbalance with the $s_C$ parameter, the generated data has the expected accuracy curves regarding minority and majority classes. Thus, we are able to generate various and proper manifestations of DC1.

### 5.3 Evaluation of Group Imbalance (DC2a)

To evaluate the presence of DC2a, we examine the imbalance of the generated groups and measure the extent of representation bias in data, i. e., if certain groups are underrepresented. To this end, we vary the $s_G$ parameter as it has the highest influence on DC2a. We use the Gini coefficient to measure the degree of imbalance for the generated groups, i. e., we apply the Gini coefficient to the group labels. Further, we report aggregated statistics about the number of instances for all groups. Figure 5 shows the minimum and the maximum number of instances of all groups as well as the Gini coefficient for the groups.

We observe an increase in the Gini coefficient for increasing $s_G$ values. For $s_G = 0$, we have a Gini coefficient of around 20%, while it is around 70% for $s_G = 2$. Thus, we are able to control the imbalance of the groups (DC2a). The maximum number of instances for each group is also increasing for higher $s_G$ values. In particular, for $s_G = 0$, we have a maximum of 84 instances for a group, while it is 398 for $s_G = 2$. On the other side, the minimum number of instances for a group is decreasing from 21 to 2. For the lowest parameter value $s_G = 0$, the Zipf distribution assigns the same probability to all groups. However, not exactly the same number of instances are assigned to all groups because it is still a random distribution. Therefore, slight deviations occur. In our approach, small deviations can occur at each level of the taxonomy due to our top-down procedure (cf. Algorithm 1), resulting in a Gini coefficient of 20% for $s_G = 0$. For the highest parameter value $s_G = 2$, some groups are underrepresented as they solely occur 2 times in the generated dataset. This shows that we can generate and control different manifestations of a representation bias in the data

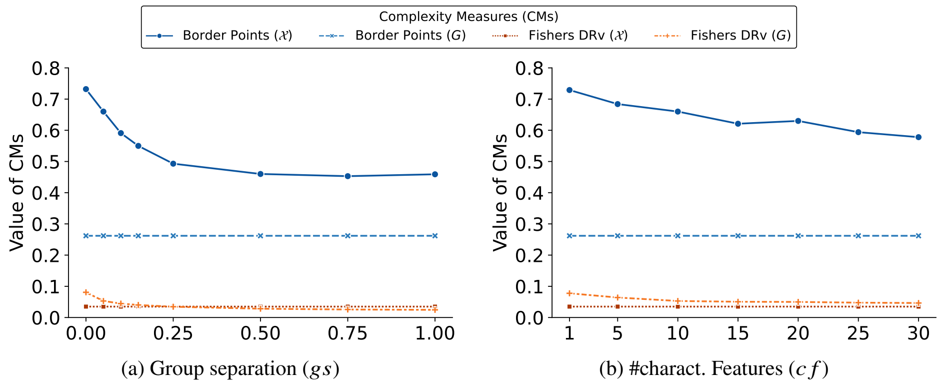


Fig. 6: Values of complexity measures for the entire data $\mathcal{X}$ and averaged over the groups G regarding (a) $gs$ and (b) $cf$.

with the $s_G$ parameter (DC2a). The real-world data from the use case of end-of-line testing have a Gini coefficient of 54% regarding the group distribution, while the rarest group has 2 instances and the most frequent group has 300 instances [HRM19, HRM20]. Thus, we are able to generate a similar group distribution with $s_G = 1.5$.

## 5.4 Evaluation of Heterogeneous Class Patterns (DC2b)

To measure the manifestation of the characteristic DC2b, we measure the effects of the aggregation bias in data (cf. Section 2). So, we focus on the difference in carrying out a data analysis (1) for the whole data and (2) for each group separately. To this end, we examine (i) the complexity of the classification problem in the generated data and (ii) the accuracy of a classification model on the generated data.

**Difference of complexity measures.** We use commonly used complexity measures [HB02] to assess the complexity of the classification problem independently of a specific classification algorithm. For sake of clarity, we focus in our discussion on the results of two commonly used complexity measures that are accompanying symptoms of the characteristics DC1 and DC2: i) The Directional-Vector Maximum Fishers Discriminant Ratio (*Fishers DRv*) measures how well the classes can be separated by the feature values. ii) The fraction of *Border Points* measures the fraction of all instances where the nearest instance belongs to a different class. We note that for both complexity measures, lower values indicate simpler classification problems, i. e., the classes are better separable or the data has less border points. We compare the complexity measures on the entire data $\mathcal{X}$ and the average values over all groups (DC2). More formally, for each of the complexity metrics $CM$, we denote the value of the $CM$ on $\mathcal{X}$ as $CM(\mathcal{X})$. We calculate the average over the groups $G = \{G_1, ..., G_k\}$ as $CM(G) = \frac{1}{k}\sum_{i=1}^{k} CM(G_i)$.

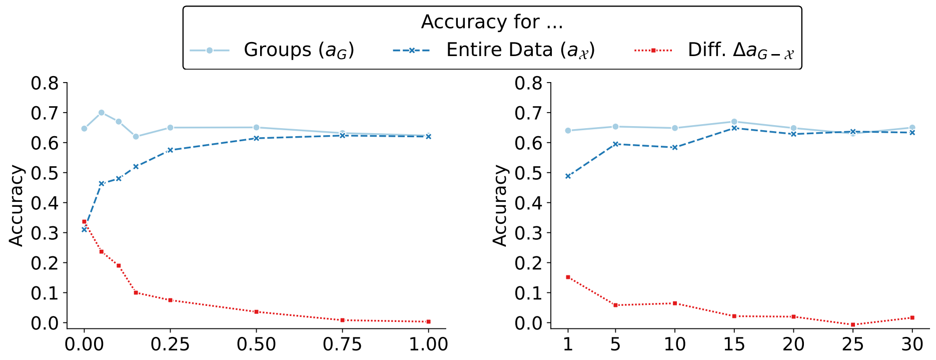


Fig. 7: Overview of accuracy results for the varying parameter configurations for our data generator. The accuracy is shown for training the model on the whole data or solely on the groups, as well as the difference in the accuracy of both.

Figure 6 shows the difference of the complexity measures averaged over all groups and on the entire data. We observe in Figure 6a that for $gs \leq 1$, the whole data $\mathcal{X}$ comprise more border points compared to the average over all groups $G$. In particular, the highest absolute difference is around 25%-points for $gs = 0$, while it is decreasing with higher $gs$ values. Thus, different classes are more often close to each other on the whole data compared to the data subsets of all groups for $gs = 0$. Nevertheless, this difference decreases for higher $gs$ values. The reason is that for $gs = 0$, the groups are not separated at all. As a result, classes from different groups have the same feature ranges and are therefore more likely to border on each other. For $gs > 0$, the groups have different value ranges for their characteristic features and thus a slight increase of $gs$ leads to a high decrease in the border points. Yet, for higher $gs$ values, the border points for $\mathcal{X}$ converge to around 45%.

We observe a similar trend for Fishers DRv, i. e., for $gs = 0$, the average value over the groups is less compared to the entire data $\mathcal{X}$. Thus, the classes are more easily separable, when considering the individual data subsets of the groups. Yet, for higher $gs$ values, Fishers DRv is decreasing on $\mathcal{X}$ and is even less than the average over the groups for $gs \geq 0.5$. Hence, the classes are better separable on $\mathcal{X}$ for $gs \geq 0.5$. This concludes that there is a similar correlation for the Fishers DRv as for the border points regarding the $gs$ value. For the $cf$ parameter (cf. Figure 6b), we observe similar trends as for $gs$. Thus, we do not discuss these results in more detail.

**Difference in accuracy.** We also measure the difference in accuracy averaged over all groups and on the entire data. To this end, similar as in Section 5.2, we train a Random Forest classifier $M_\mathcal{X}$ on $\mathcal{X}_{train}$ and report the accuracy on the test set $\mathcal{X}_{test}$ as $a_\mathcal{X}$. Further, we train a set of classifiers $\mathcal{M}_G = \{M_1, ..., M_g\}$, where each $M_i$ is a Random Forest classifier trained separately on a group $G_i \subset \mathcal{X}_{train}$. In this case, we predict the class label for each test instance $x_{test} \in \mathcal{X}_{test}$ that belongs to group $G_j$ with the model $M_j$. Thus, we denote the accuracy of $\mathcal{M}_G$ for $\mathcal{X}_{test}$ with $a_G$. Further, we define the difference in accuracy as $\Delta a_{G-\mathcal{X}} = a_G - a_{\mathcal{X}}$.

Figure 7 shows the accuracy results. For the $gs$ parameter (cf. Figure 7a), we observe that the highest difference in accuracy is obtained for $gs = 0$, which is more than 30%-points. Thus, we have the strongest effect of an aggregation bias for $gs = 0$. Yet, the difference in the accuracy is decreasing for higher $gs$ values, i. e., for $gs = 1$ it is about 5%-points. The reason can be seen in the previous results for the complexity measures: The data has less border points and the classes are more separable on $\mathcal{X}$ for higher $gs$ values. Thus, it is easier for a classification model to predict the classes more accurately on $\mathcal{X}$. Again, we observe similar results for the $cf$ parameter in Figure 7b. We note that Random Forest achieves an accuracy of 33% on real-world data that comprise both characteristics [HRM19, HRM20]. Thus, we can generate data with similar accuracy results using $gs = 0$.

Concluding, the results show that we are able to control the difference of the complexity measures and the accuracy with the $gs$ and the $cf$ parameters. In other words, our data generator is able to control the heterogeneity of the class patterns, i. e., the aggregation bias in the generated data (DC2b).

## 6 Conclusion

The contribution of this paper is an approach to generate synthetic data comprising two data characteristics that often occur in real-world use cases: multi-class imbalance (DC1) and heterogeneous groups (DC2). The actual manifestations of these data characteristics are domain-specific, i. e., dependent on the actual real-world use case. Therefore, our approach uses a taxonomy model and a two-step process to generate data that reflect the characteristics of a given real-world use case. A taxonomy is the simplest form of knowledge model to organize real-world entities in domain-specific groups and it can be found in various domains. So, our approach is not limited to a specific domain. In our evaluation, we unveil that the generated data comprises the characteristics DC1 and DC2 together. Moreover, the parameters of our data generator may be steered to reflect different manifestations of these characteristics. Our approach builds the fundamental basis for future work to create a benchmark that evaluates machine learning and data engineering approaches systematically on data with the characteristics DC1 and DC2. Thereby, correlations between different manifestations of the characteristics and the performance of approaches can be examined.

**Acknowledgements.** This research was performed in the project 'VALID-PARTITION' as part of the Software Campus program, which is funded by the German Federal Ministry of Education and Research (BMBF) under Grant No.: 01IS17051. This work was also supported by the Ministry of Science, Research, and the Arts of the State of Baden-Wurttemberg within the sustainability support of the projects of the Excellence Initiative II.

## Bibliography

[AK04] Agard, Bruno; Kusiak, A.: Data-Mining-based Methodology for the Design of Product Families. International Journal of Production Research, 42(15):2955–2969, 2004.

[Al11] Alcalá-Fdez, Jesús et al.: KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. Multiple-Valued Logic and Soft Computing, 2011.

[Ch20] Chan, Stephanie et al.: Machine Learning in Dermatology: Current Applications, Opportunities, and Limitations. Dermatology and Therapy, 2020.

[Ci09] Cimiano, Philipp et al.: Ontology learning. In: Handbook on ontologies, pp. 245–267. Springer, 2009.

[Co00] Cowell, Frank: Measuring Inequality. $3^{\text{th}}$ edition, 2000.

[DC19] Dahmen, Jessamyn; Cook, Diane: SynSys: A synthetic data generation system for healthcare applications. Sensors, 19(5):1181, 2019.

[DG17] Dua, Dheeru; Graff, Casey: , UCI Machine Learning Repository, 2017.

[Fe13] Fernández, Alberto et al.: Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches. Knowledge-Based Systems, 42:97–110, apr 2013.

[Fe20] Fernandes, Ederson Carvalhar et al.: Flexible Production Data Generator for Manufacturing Companies. Procedia Manufacturing, 2020.

[Fr11] Frasch, Janick V. et al.: A Bayes-true data generator for evaluation of supervised and unsupervised learning methods. Pattern Recognition Letters, 32(11):1523–1531, 2011.

[FS18] Fränti, Pasi; Sieranoja, Sami: K-means properties on six clustering benchmark datasets. Applied Intelligence, 48:4743–4759, 2018.

[Ga12] Galar, Mikel et al.: A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42(4):463–484, 2012.

[GBC16] Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron: Deep learning. MIT press, 2016.

[Gr22] Gröger, Christoph: Industrial Analytics — An Overview. it – Information Technology, 64(1–2):55–65, 2022.

[Gu03] Guyon, Isabelle: Design of experiments for the NIPS 2003 variable selection benchmark. 2003.

[HB02] Ho, Tin Kam; Basu, Mitra: Complexity measures of supervised classification problems. IEEE TPAMI, 2002.

[HG09] He, Haibo; Garcia, E.A.: Learning from Imbalanced Data. IEEE TKDE, 21(9):1263–1284, sep 2009.

[Ho21] Hogan, Aidan et al.: Knowledge graphs. Synthesis Lectures on Data, Semantics, and Knowledge, 12(2):1–257, 2021.

[HRM19] Hirsch, Vitali; Reimann, Peter; Mitschang, Bernhard: Data-driven fault diagnosis in end-of-line testing of complex products. In: IEEE DSAA. 2019.

[HRM20] Hirsch, Vitali; Reimann, Peter; Mitschang, Bernhard: Exploiting domain knowledge to address multi-class imbalance and a heterogeneous feature space in classification tasks for manufacturing data. VLDB, 2020.

[Ig19] Iglesias, Félix et al.: MDCGen: Multidimensional Dataset Generator for Clustering. Journal of Classification, 2019.

[Ja04] Jablonski, Nina: The Evolution of Human Skin and Skin Color. Ann. Review of Anthropology, 33:585–623, 2004.

[KBT11] Köksal, Gülser; Batmaz, İnci; Testik, Murat Caner: A review of data mining applications for quality improvement in manufacturing industry. Expert Systems with Applications, 38(10):13448–13467, sep 2011.

[KM16] Kassner, Laura; Mitschang, B.: Exploring Text Classification for Messy Data: An Industry Use Case for Domain-Specific Analytics. In: EDBT. 2016.

[KRM19] Kiefer, Cornelia; Reimann, Peter; Mitschang, Bernhard: A Hybrid Information Extraction Approach Exploiting Structured Data Within a Text Mining Process. In: BTW 2019. Gesellschaft für Informatik, Bonn, pp. 149–168, 2019.

[KU15] Khan, Aunsia; Usman, Muhammad: Early Diagnosis of Alzheimer's Disease Using Machine Learning Techniques: A Review Paper. In: Proc. of the $7^{\text{th}}$ International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K). Lisbon, Portugal, pp. 380–387, 2015.

[LKJ02] Lundin, Emilie; Kvarnström, Håkan; Jonsson, Erland: A Synthetic Fraud Data Generation Methodology. In: Information and Communications Security. 2002.

[Me21] Mehrabi, Ninareh et al.: A Survey on Bias and Fairness in Machine Learning. ACM Comput. Surv., 54(6), jul 2021.

[MGTM20] Maitín, Ana María; García-Tejedor, Alvaro José; Muñoz, Juan Pablo Romero: Machine Learning Approaches for Detecting Parkinson's Disease from EEG Analysis: A Systematic Review. Applied Sciences, 10(23), 2020.

[PWV16] Patki, Neha; Wedge, Roy; Veeramachaneni, Kalyan: The Synthetic Data Vault. In: IEEE DSAA. 2016.

[RHW21] Roh, Yuji; Heo, Geon; Whang, Steven Euijong: A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective. IEEE TKDE, 33(4):1328–1347, apr 2021.

[SB21] Steinbuss, Georg; Böhm, Klemens: Benchmarking Unsupervised Outlier Detection with Realistic Synthetic Data. ACM TKDD, 2021.

[Sc12] Schervish, Mark J: Theory of statistics. Springer Science & Business Media, 2012.

[SG21] Suresh, Harini; Guttag, John: A Framework for Understanding Sources of Harm throughout the Machine Learning Life Cycle. In: Equity and Access in Algorithms, Mechanisms, and Optimization. EAAMO '21, 2021.

[SGG18] Suresh, Harini; Gong, Jen J.; Guttag, John V.: Learning Tasks for Multitask Learning: Heterogenous Patient Populations in the ICU. In: SIGKDD. 2018.

[SH05] Steinley, Douglas; Henson, Robert: OCLUS: An Analytic Method for Generating Clusters with Known Overlap. Journal of Classification, 2005.

[So91] Sowa, John F.: Principles of Semantic Networks. Explorations in the Representation of Knowledge. Morgan Kaufmann, 1991.

[Su14] Sun, Chong et al.: Chimera: Large-Scale Classification using Machine Learning, Rules, and Crowdsourcing. VLDB, 2014.

[SWK09] Sun, Yanmin; Wong, Andrew; Kamel, Mohamed: Classification of Imbalanced Data: A Review. International Journal of Pattern Recognition and Artificial Intelligence, 23(4):687–719, 2009.

[ULP19] Ustun, Berk; Liu, Yang; Parkes, David: Fairness without harm: Decoupled classifiers with preference guarantees. In: ICML. 2019.

[Va14] Vanschoren, Joaquin et al.: OpenML: Networked Science in Machine Learning. SIGKDD Explor. Newsl., 15(2):49–60, jun 2014.

[Wi20] Wilhelm, Yannick et al.: Data Science Approaches to Quality Control in Manufacturing: A Review of Problems, Challenges and Architecture. In: Proc. of the 14th Symposium on Service-Oriented Computing (SummerSOC). Communications in Computer and Information Science (CCIS). Springer-Verlag, pp. 45–65, 2020.

[WLL21] Wang, Jialu; Liu, Yang; Levy, Caleb: Fair Classification with Group-Dependent Label Noise. In: Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency. FAccT ’21, Association for Computing Machinery, New York, NY, USA, p. 526–536, 2021.

[Wu16] Wuest, Thorsten et al.: Machine learning in manufacturing: advantages, challenges, and applications. Production & Manufacturing Research, 4(1):23–45, jan 2016.

[WY12] Wang, Shuo; Yao, Xin: Multiclass Imbalance Problems: Analysis and Potential Solutions. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 42(4):1119–1130, 2012.