FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Binary Data Balancing Methods
Student:	Bc. Michaela Kučerová
Supervisor:	Ing. Magda Friedjungová, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2023/2024

Instructions

Imbalanced data typically refers to a problem with classification tasks where the classes are not represented equally. One of the possible ways how to solve this problem is to use data-level methods to generate artificial training data to balance the dataset. The aim of this thesis is to experimentally compare several balancing methods.

1. Survey common (such as SMOTE) and state-of-the-art oversampling algorithms (for example based on generative adversarial networks (GAN)) for tabular data balancing. Focus on binary classification.
2. Use an existing implementation or implement at least five of the reviewed approaches. Consider usage of SMOTE, VAE and GAN-based methods.
3. Experimentally compare their performance (classification accuracy and F1 score) on publicly available datasets frequently used in papers reviewed in Step 1. Comparison of data balancing methods will be done using a classification model learned on both, original (unbalanced) and synthetically balanced data.
4. Design and implement your own method to generate synthetical tabular data. Experimentally compare its performance as described in Step 3.
5. Discuss the results obtained and the benefits and limitations of the algorithms used.

Electronically approved by Ing. Karel Klouda, Ph.D. on 15 November 2022 in Prague.

Master's thesis

BINARY DATA BALANCING METHODS

Bc. Michaela Kučerová

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing. Magda Friedjungová, Ph.D.
January 11, 2024

Contents

List of Figures

List of Tables

First of all, I would like to thank my supervisor, Ing. Magda Friedjungová, Ph.D., for her guidance and valuable advice throughout the whole writing process. I would also like to express my gratitude to the entire Faculty of Information Technology for the excellent study programme I was lucky to absolve. My thanks also go to my family and friends who supported me, believed in me, and gave me strength to reach this point.

Declaration

In Prague on January 11, 2024

Abstract

Many real-world tabular datasets are imbalanced, which has a negative effect on the quality of the models applied to those data. More ways exist to deal with this problem, and one of them is re-balancing the dataset. This master's thesis presents a review of existing oversampling methods, such as SMOTE, for imbalanced datasets of binary classification problem. Traditional methods, as well as generative-based techniques, are outlined in this work. Moreover, a novel oversampling method called LIT-GAN is presented. LIT-GAN combines an interpolation technique with generative models. Selected ten oversampling methods and the novel method are compared experimentally using classification on frequently used datasets with six different evaluation metrics.

Keywords binary classification, oversampling, data balancing, tabular data, class imbalance

Abstrakt

Spousta tabulkových datasetů reálného světa je nevyvážená, což má negativní dopad na kvalitu výstupů modelů, jež používají tato nevyvážená data. Existuje řada přístupů, jak se s tímto problémem vypořádat, přičemž jedním z nich je balancování datasetu. Tato magisterská práce obsahuje přehled existujících balančních metod, jako je například SMOTE, pro nevyvážené datasety problému binární klasifikace. V práci jsou nastíněny tradiční metody i techniky založené na generativních metodách. Kromě toho je představena nová metoda vzorkování nazvaná LIT-GAN. Tato metoda kombinuje techniku interpolace s generativními modely. Vybraných deset metod vzorkování a tato nová metoda LIT-GAN jsou experimentálně porovnány pomocí klasifikace na často používaných datasetech se šesti evaluačními metrikami.

Klíčová slova binární klasifikace, vzorkování, balancování dat, tabulková data, nevyváženost tříd

List of abbreviations

AAE	Adversarial Autoencoder
AC	Auxiliary Classifier
ACAI	Adversarially Constrained Autoencoder Interpolation
ADASYN	Adaptive Synthetic
AE	Autoencoder
AUC	Area Under The ROC Curve
BA	Balanced Accuracy
CBO	Cluster-Based Oversampling
CNN	Convolutional Neural Network
CTAB-GAN	Conditional Table Generative Adversarial Network
CTGAN	Conditional Tabular Generative Adversarial Network
cWGAN	Conditional Wasserstein GAN
DCGAN	Deep Convolutional Generative Adversarial Network
DTC	Decision Tree Classifier
ENN	Edited Nearest Neighbors
GAN	Generative Adversarial Network
GBC	Gradient Boosting Classifier
GDPR	European General Data Protection Regulation
GMM	Gaussian Mixture Model
HP	Hyperparameter
IAAE	Interpolate Adversarial Autoencoder
IDS	Imbalanced Dataset
IR	Imbalance Ratio
kNN	k Nearest Neighbors
KNN	k Neighbors Classifier
LIT-GAN	Latent Interpolation Tabular Generative Adversarial Network
LOGIT	Logistic Regression
LoRAS	Localized Random Affine Shadowsampling
LSTM	Long-Short-Term Memory
ML	Machine Learning
MLP	Multi-Layer Perceptron
MedGAN	Medical Generative Adversarial Network
MGVAE	Majority-Guided Variational Autoencoder
MWMOTE	Majority Weighted Minority Oversampling Technique
OA	Overall Accuracy
OCC	One-class Classification
PL	Proximity Level
Poly	Polynom-Fit-SMOTE
ProWSyn	Proximity Weighted Synthetic Oversampling Technique
RAMOBoost	Ranked Minority Oversampling in Boosting
RAE	Regularized Autoencoder
RFC	Random Forest Classifier
ROC	Receiver Operating Characteristics
ROS	Random Oversampling
SDE	Stochastic Differential Equation
SGM	Score-based Generative Model
SMOTE	Synthetic Minority Oversampling Technique
SOS	Score-based OverSampling
SVM	Support Vector Machine
TAEI	Tabular Autoencoder Interpolation
TTGAN	Tabular Translation Generative Adversarial Network
VAE	Variational Autoencoder
VGM	Variational Gaussian Mixture Model

Introduction

"What I dream of is an art of balance."

- Henri Matisse, 1869–1954

Data are everywhere. In recent years, people have produced enormous amounts of data in their everyday lives that can be used to gain knowledge and can help to make decisions. However, not all of them are adequate for such a usage. The data must be stored correctly; if the primary use is making a classification decision, the data should be ideally balanced. However, we live in a world where not everything is ideal. For example, real-world datasets can be imbalanced, which can cause problems when trying to gain knowledge using the data. The problem of imbalanced datasets has been researched for more than two decades. There have been conferences and workshops concentrating on this topic, such as AAAI'2000 Workshop on Learning from Imbalanced Data Sets[1] and ICML'2003 Workshop on Learning from Imbalanced Data Sets (II)[2] and from the recent years, Classifier Learning from Difficult Data: CLD^{2}2019[3].

Several approaches exist to overcome the imbalanced dataset (IDS) problem, such as re-balancing the dataset or adjusting the model. The goal of this thesis is to survey standard methods dealing with this problem by oversampling and implement at least five of those methods. Furthermore, the goal is to use a classifier learned on the original (imbalanced) dataset and datasets balanced by adding synthetic samples produced by the oversampling algorithms and compare those performances. Additionally, this work aims to introduce a novel data-balancing approach and compare its performance to the standard oversampling algorithms.

This thesis focuses on oversampling tabular data for binary classification. However, the topic of imbalanced datasets is broad. There exist also other types of datasets, such as image or time series datasets that can be imbalanced and need to be oversampled and handled differently. What is more, other approaches than oversampling for dealing with this problem exist. However, those other domains and approaches are not the subject of this thesis.

[1]`https://www.site.uottawa.ca/~nat/Workshop2000/workshop2000.html`
[2]`https://www.site.uottawa.ca/~nat/Workshop2003/workshop2003.html`
[3]`http://cldd.kssk.pwr.edu.pl/`

Chapter 1

Background

In this Chapter, the fundamentals related to this thesis topic are introduced. Firstly, the binary classification problem and tabular datasets are presented. Secondly, the problem of learning from an imbalanced dataset is outlined.

1.1 Binary Classification

This work focuses on binary classification, which deals with the problem of assigning each data sample a label Y based on its characteristics. The characteristic is given by values of features $X_0, X_1, ...X_n$. We are looking for a classifier f, for which it holds:

$$Y \approx f(X_0, X_1, \ldots, X_n)$$

Different classification models exist, and some are introduced in Section 4.3. The classifier's complexity depends on the label Y. [1]

In the case of binary classification, variable Y can take only two values, for example, $0, 1$ or in the case of images, using binary classification, it can say if there is a man or a woman in the image. However, the main goal is the same: construct a model that best predicts the value of Y.

1.2 Tabular Dataset

In Section 1.1, the binary classification problem is introduced, which is directly related to the variable Y. However, nothing is said there about the features $X_0, X_1, \ldots, X_n$. Those features can differ in their data types.

In the case of binary images, features represent individual pixels and take values in a given range. Their order is also important. On the other hand, a tabular dataset refers to data that can be stored in a table with rows and columns. Each column of this table represents one feature, and each row represents one sample[1]. What is more, the order of the columns is not essential. This thesis focuses exclusively on tabular datasets.

Tabular data are common in the real world. For example, each computer folder can represent tabular data, with each file being a sample and its name, creation date, file type, etc., representing features. Another example can be a list of personal data, having a name, age, sex, etc., as features for each item, where one personal data represent one observation.

Each feature in a tabular dataset has a specified domain. The domains can be either quantitative or qualitative [2].

[1]In this work, *sample* and *observation* are used interchangeably.

Qualitative data cannot be measured the same as numbers. They can be subdivided into two types: nominal and ordinal.

Nominal data types do not have an order. An example of a nominal data type is a movie genre *(fantasy, thriller, comedy,...)*. Dichotomous data types are special nominal types with only two values. An example is a gender with values *man* and *woman*. Ordinal data types are nominal data types with an order of their values but without specified distance between them. Examples of ordinal data types are a T-shirt size *(XS, S, M, L)* and a grade *(A, B, C, D, E, F)*.

Quantitative data types, on the other hand, represent numerical types. They can be counted, and a difference between each two values is defined. Quantitative data types are further subdivided into two types: discrete and continuous.

Discrete data are represented by whole numbers. They are countable and cannot be further subdivided. Their values are also finite. Examples of discrete data types are age or number of students in a class. Fractional numbers represent continuous data. They can be further divided and can have any value in a range. Examples of continuous data are height, temperature, or speed. From the storage point of view, discrete data are stored as integers, whereas continuous data are stored as floats or doubles.

Generating tabular data raises some challenges that differ from generating images. Tabular data can have various data types of features, each having to be handled individually.

1.3 Imbalanced Data

Having an imbalanced dataset is a common issue in science, industry, and everyday life. An imbalanced dataset is one where one class is represented by a much smaller number of instances than the other class [3]. The first class is called the minority or also the positive class, and the second one is called the majority or negative class.[2] Some examples where the problem of having imbalanced data occurs are medicine, fraud detection, bioinformatics, manufacturing process failures, or anomaly detection [4], [3], [5].

Having such an imbalanced dataset means a problem for a classification model to identify cases of interest, which are usually the minority class samples. The classifiers can be biased towards majority class recognition because they aim to achieve overall accuracy (OA) [5]. In medicine, for example, the main focus can be to identify if a person suffers from a disease or not. However, if the disease is rare, it represents the minority class in a dataset with a label saying if the person has or does not have the disease because the number of healthy people outnumbers the ill ones.

In the work by Visa et al. [5], the authors distinguish two main components of imbalanced datasets. Those are the imbalance ratio (IR) and the lack of information (LI) for the minority class. Imbalance ratio is defined as a fraction of the number of majority class samples (N_-) and the number of minority class samples (N_+) in a dataset [6]:

$$IR = \frac{N_-}{N_+}.$$

Two datasets having the same IR can have a different LI. The dataset suffers more from the lack of information when there are only a few minority samples.

However, the problem of an imbalanced dataset is not only having too few samples in the minority class. If the two classes are separable and have a clear decision boundary, there is no need to modify the training dataset by balancing. Therefore, the problem arises if the classes are overlapping. In the overlapping areas, for example, using neural networks, the minority samples can be considered noise and, therefore, even discarded [5]. What is more, the complexity of the data plays a significant role in the classification performance of a model trained on this dataset.

[2] Whenever sign $+$ or $-$ appears by an observation or a set of samples, it denotes belonging to positive, respectively negative class.

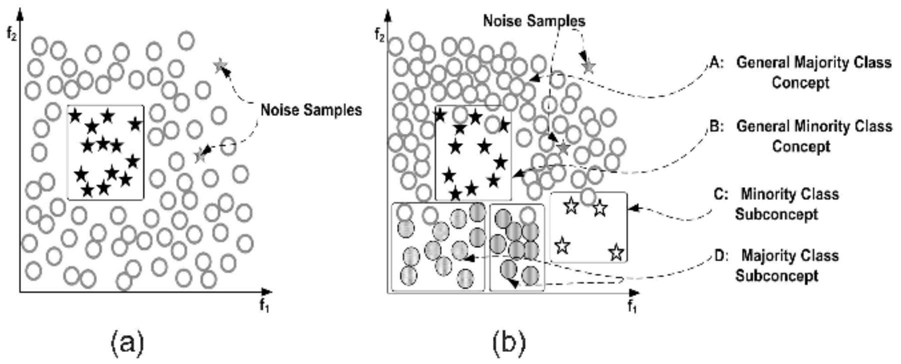

f2
f1
Noise Samples
(a)
Noise Samples
A: General Majority Class Concept
B: General Minority Class Concept
C: Minority Class Subconcept
D: Majority Class Subconcept
(b)

Figure 1.1 IDS Concepts taken from [7]: (a) between-class imbalance and noise (b) between-class imbalance, within-class imbalance, overlapping, and noise

In the article by Lemnaru et al. [3], the problem of IDS is described as having two different components. Those are insufficient data for model building and many special cases in the minority class. The first issue concerns LI about the minority class, and the second is the problem of considering minority samples as noise. In the latter case, subclusters occur. Subclusters are subdivisions of a whole that combine elements to form a smaller group. In Section 2.1.3, some methods dealing with this particular problem are introduced.

Having subclusters is directly connected to the problem of IDS, where the imbalance can occur not only between classes, as described above, but also within each one. Those two types of class imbalance are called between-class and within-class imbalance. The latter occurs when the class samples exist in subclusters with a limited number of instances in some of them. For those subclusters, it is hard for the model to learn rules for their correct classification. [7]

The problem of imbalanced learning from tabular data is rather complex. The class instances appear in subclusters that can be sparse or dense, noise observations can also appear in the dataset, and classes can overlap. Moreover, both between-class and within-class imbalances can occur in the dataset. All those concepts are presented in Figure 1.1.

1.3.1 Approaches

Several methods for dealing with the IDS problem exist, and they can be divided into three groups of approaches: data-level, algorithm-level, and hybrid [8].

The data-level approach can be subdivided into undersampling and oversampling methods, where both modify the training set [9]. The main idea of undersampling is removing samples of the majority class. This can be done until the number of majority class samples matches the number of minority class samples. Oversampling, on the other hand, is based on adding synthetic samples into the minority class. The advantage of oversampling compared to undersampling is no loss of information. However, oversampling can lead to the creation of meaningless synthetic samples. The advantage of re-sampling methods is that they are part of preprocessing, and therefore, they do not affect the training and do not require a specific model to be used for downstream tasks. They can be applied to minority class samples, majority class samples, or both. [10]

The algorithm-level approach, in contrast, concentrates on modifying the model [9]. Those methods require insight into the algorithm because it has to be known why it performs poorly and what has to be changed [8]. One possible way is to adjust the algorithm's weights. The minority class samples are assigned a higher weight in the cost function than the majority class samples. This results in bigger penalization when the minority class observations are misclassified. Another solution is applying a one-class classification that concentrates only on the target group, thus eliminating bias towards one class [9]. The classifier is fitted on the majority class, and the minority class samples are considered outliers. Therefore, this method is also used for anomaly and outlier detection.

The third approach is a hybrid one, and it represents a mix of the previous two approaches to extract the advantages of both and reduce their weaknesses. One example is combining sampling and cost-sensitive learning [8]. In cost-sensitive learning, each observation is assigned a misclassification cost and the aim is to minimize the total cost [11]. This idea of merging data-level and algorithm-level approaches results in having a robust model [9].

This thesis concentrates on the data-level oversampling approach for the tabular data binary classification problem. However, the whole overview is outlined here to give an idea of how complex the IDS problem is and what the possibilities are to deal with it.

Chapter 2
Existing Methods

This chapter contains an overview of standard oversampling algorithms, which can be subdivided based on their approach and structure. Some of the most used algorithms are introduced for each of those approaches.

The problem of an imbalanced dataset has been studied for over two decades; therefore, many oversampling approaches exist to deal with this problem. The algorithms are divided into three parts: traditional, generative, and their combination. Traditional methods are the older ones that first appeared more than twenty years ago. On the other hand, generative methods became popular in recent years with the popularity of deep learning, especially the introduction of Generative Adversarial Network (GAN) [12].

2.1 Traditional Oversampling Algorithms

In the work by Kovács [6], 85 minority oversampling methods are compared, supporting the claim that the IDS problem is focused on by many researchers who invent new, improved methods.

The first presented and the most naive algorithm is Random Oversampling (ROS). ROS does the oversampling by randomly selecting minority class samples and creating exact duplicates of them. The problem with this algorithm is that instead of generalizing the decision boundary, it becomes smaller and more specific and, therefore, can lead to overfitting. [13]

2.1.1 Synthetic Data Generation

Using synthetic data generation, new samples, not only duplicates, are created. Compared to ROS, this approach can create a more generalized decision boundary and improve classification performance.

2.1.1.1 SMOTE

Many oversampling methods exist based on the Synthetic Minority Oversampling TEchnique (SMOTE) [14]. SMOTE can be further combined with boosting, kernel-based learning, or cleaning noisy samples.

SMOTE is one of the oldest minority oversampling techniques. It was introduced in 2002 by Chawla et al. [14]. In contrast to ROS, which duplicates samples, SMOTE creates synthetic samples rather than duplicates. The idea was taken from an article about handwritten character recognition, where original data were modified in order to generate more training data.

SMOTE algorithm takes each minority sample and its k nearest neighbors (kNN) in the feature space. From those kNN samples, one is chosen randomly, and a synthetic sample is generated on the line connecting the original sample and the selected neighbor. The number of selected neighbors for oversampling depends on the amount of synthetic data to be oversampled. The samples are generated as follows: “Take the difference between the feature vector (sample) under consideration and its nearest neighbor. Multiply this difference by a random number between 0 and 1 and add it to the feature vector under consideration.” [14]

$$x_{syn} = \delta \cdot (x_{train} - x_{neighbor}) + x_{train}, \quad \delta \in [0, 1] \tag{2.1}$$

This leads to the generalization of the decision boundary contrary to ROS. However, minority class samples may disturb the space of the majority class. In that case, it can be useful to combine SMOTE with data cleaning methods like Tomek links or Edited Nearest Neighbors (ENN) [15].

SMOTE-NC
Synthetic Minority Oversampling Technique-Nominal Continuous is a SMOTE extension designed for datasets with nominal and continuous features. This algorithm takes all minority class samples and counts the median of their standard deviations for all continuous features. This value is added to the Euclidean distance of the selected sample and its neighbor if they differ in the value of a nominal feature (one addition for each difference). The value of a nominal feature of the synthetic sample is selected by the majority of its kNN. [14]

SMOTE-N
SMOTE-NC works only for data with both continuous and nominal features. SMOTE-Nominal, as the name says, is another SMOTE extension that can be applied to datasets with only nominal features [14].

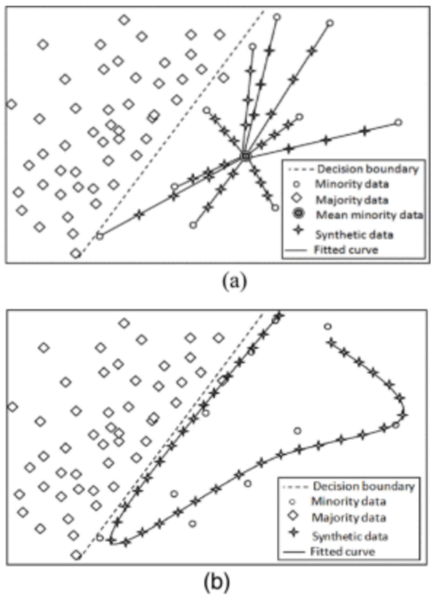
However, SMOTE does not consider the majority class samples. It generates the same amount of synthetic samples for each minority sample. The problem with this algorithm is that some observations are harder to classify than others, but SMOTE does not compensate for skewed distributions [7]. To solve this problem, adaptive oversampling methods operating mainly near the decision boundary were introduced. Some of those methods are outlined in Section 2.1.2.

Furthermore, SMOTE is a widely used oversampling technique also used as a baseline in many research works for comparison. In an article marking its 15-year anniversary [16], an overview of dozens of SMOTE-based methods is presented. This overview can be seen in Appendix A in Figures A.1 and A.2 that show more than 85 SMOTE algorithm extensions and 17 SMOTE-based ensemble methods. The authors presented these as an overview instead of a detailed description because of space limitations for their article. However, the essential characteristics are captured in those Figures.

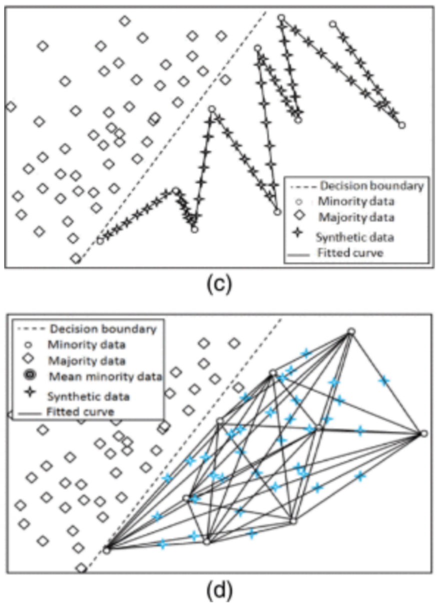
2.1.1.2 Polynom-Fit-SMOTE

Polynom-Fit-SMOTE (Poly) [10] represents a SMOTE extension. It can use four different polynomial fitting functions based on their topologies: Star, Mesh, Bus, and Polynomial curved-bus. “The method operates in feature space and generates new samples using Curve Fitting methods to find the coefficients of a polynomial *p(x)* of degree n that fits the minority class.” [10]

The names of the topologies denote the way synthetic samples are generated. The topologies can be found in Figure 2.1. For example, generated samples are arranged into a star using the Star topology. Having the Mesh topology, connections to all other minority samples from the selected sample are used to generate a new observation. Based on the original article [10], those two topologies perform the best. In the work by Kovács [6], Poly proved to be the best overall oversampling technique from 85 SMOTE extensions that have been used.



(a) Star and Polynomial curved-bus



(b) Bus and Mesh

Figure 2.1 Polynom-Fit-SMOTE taken from [10]: Topologies

2.1.1.3 ProWSyn

Proximity Weighted Synthetic Oversampling Technique (ProWSyn) [17] is an oversampling method that creates weights for minority class samples, which are assigned based on their distance from majority class samples. This is the first introduced method using information from the majority class for minority class oversampling. In the work by Kovács [6], this algorithm appeared to be the second best overall. The minority class samples are divided into groups based on their proximity level (PL), given by the distance from the majority class samples.

The algorithm searches for `L` partitions with increasing PL. In each of the `L` loops, it searches for each majority class sample x_- for its kNN of the minority class from `P`, where `P` is initialized as the set of all minority class samples. All those neighbors are unioned and create a partition with PL assigned a value of the loop step i: $P_i = i$. Those minority class samples belonging to the current partition P_i are removed from `P`, and the algorithm proceeds with another round.

After all minority class samples are assigned a partition with a PL, the weights are counted for each partition j belonging to PL_j as:

$$w_j = exp(-\theta \cdot PL_j - 1)$$

where θ is used for smoothing. The weights are normalized for each j to get $\hat{w_j}$, and the number of samples to be generated for each PL_j is counted based on those weights.

The minority class samples are then clustered, and for each minority sample x_i, samples from its cluster are randomly selected and synthetic samples are generated the same way as in 2.1.

2.1.1.4 LoRAS

"Localized Random Affine Shadowsampling (LoRAS) oversamples from approximated data manifold of the minority class." [18] This oversampling technique was introduced in 2020 and confirms that traditional methods are not just a part of history but are still being developed and used. The synthetic samples are taken from the approximation of the minority class data manifold.

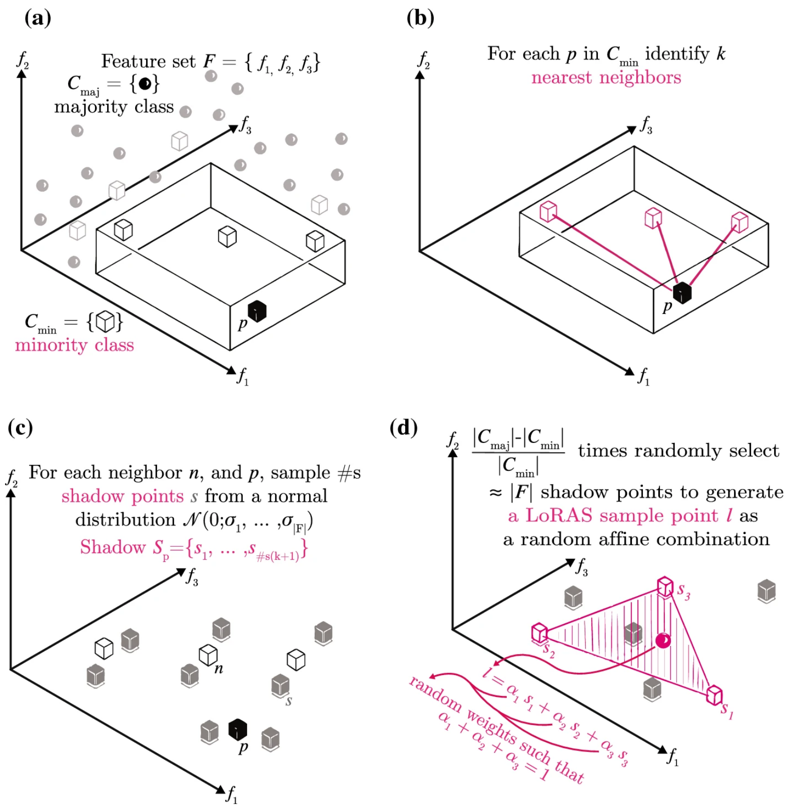

(a)
Feature set $F = \{f_1, f_2, f_3\}$
$C_{maj} = \{●\}$
majority class
$C_{min} = \{⬡\}$
minority class
p
(b)
For each p in C_{min} identify k
nearest neighbors
(c)
For each neighbor n, and p, sample #s
shadow points s from a normal
distribution $\mathcal{N}(0;\sigma_1, \ldots, \sigma_{|F|})$
Shadow $S_p=\{s_1, \ldots, s_{\#s(k+1)}\}$
(d)
$\frac{|C_{maj}|-|C_{min}|}{|C_{min}|}$ times randomly select
$\approx |F|$ shadow points to generate
a LoRAS sample point l as
a random affine combination
$l = \alpha_1 s_1 + \alpha_2 s_2 + \alpha_3 s_3$
random weights such that
$\alpha_1 + \alpha_2 + \alpha_3 = 1$
f_1
f_2
f_3

Figure 2.2 LoRAS taken from [18]

SMOTE uses two minority class samples and creates a new sample as their convex combination. LoRAS, on the other hand, generates noisy samples (shadowsamples) from Gaussian noise in the neighborhood of a minority sample, and the final synthetic sample (Localized Random Affine Shadowsample) is created as a random affine combination of those shadowsamples.

The steps for each minority sample x are the calculation of its kNN, then, for each sample in the kNN, retrieval of shadowsamples from the normal distribution $\mathcal{N}(0, h(\sigma_f))$, where $h(\sigma_f)$ is some function of the sample variance σ_f for feature $f \in F$. This is done for all features. The next step is LoRAS sample generation from a randomly selected fixed number of shadowsamples multiplied by random normalized weights (affine weights). The idea is demonstrated in the Figure 2.2.

According to the original article [18], LoRAS can, for each minority sample, better estimate the mean of the underlying local distribution than other techniques like SMOTE.

2.1.2 Borderline Oversampling Methods

These techniques do not generate new samples uniformly or randomly for each minority class observation. They try to identify the samples near the decision boundary and sample more in those regions around hard-to-learn samples.

2.1.2.1 Borderline-SMOTE

Borderline-SMOTE [13] is a SMOTE extension that deals with the problem of different classification difficulties of minority class samples. The idea is to synthesize new samples more around minority observations located near the decision boundary. In the work by Han et al. [13], two versions Borderline-SMOTE1 and Borderline-SMOTE2 were presented.

The algorithm works like this: for each minority sample (target), its kNN are found. If all of them are from the majority class, the target is considered to be a noise. If more than half of the neighbors are majority class samples, the target is added to a set called `DANGER`. For all samples in `DANGER`, kNN from minority samples are found, and the algorithm proceeds further as SMOTE (Eq. 2.1).

This is how Borderline-SMOTE1 works. Borderline-SMOTE2 differs in the kNN searching step for each sample in the `DANGER` set. The neighbors are found not only in the minority class samples but also in the majority class samples. For those, the distance between the target and its neighbor is multiplied by a random number between 0 and $\frac{1}{2}$ ($\delta \in [0, \frac{1}{2}]$) instead of $[0, 1]$. This forces the synthesized sample to be created closer to the target, in a safer region.

2.1.2.2 ADASYN

Adaptive Synthetic (ADASYN) [19] oversampling is an adaptive method introduced by Shutao Li in 2008. This algorithm concentrates on the class distributions. The synthetic samples are not generated uniformly for each minority sample, but more samples are generated for those that are hard to classify.

For each minority sample x_i, its kNN are found, and the ratio r_i of majority class samples between them is computed. The ratios are normalized to $\hat{r_i}$, and the number of synthetic samples to be generated for each of the minority class samples (g_i) is proportional to the normalized ratio:

$$g_i = \hat{r_i} \cdot G$$

where G is the total number of samples to be generated, counted as:

$$G = (N_- - N_+) \cdot \beta, \quad \beta \in [0, 1].$$

The new samples are generated in the same way as for SMOTE. Only kNN of the minority class are considered for each x_i, g_i of those are randomly chosen, and new samples are created on the line connecting x_i and the neighbor as in Eq. 2.1.

2.1.2.3 Safe-Level-SMOTE

Safe-Level-Synthetic Minority Oversampling Technique [20] deals with the problem of SMOTE: not considering the majority class samples. It assigns each of the minority class observations a safe level and generates new samples based on those values. The safe level is defined as the number of positive instances in the kNN for each positive sample.

For each minority sample x with safe level sl_x, one of its kNN denoted as n with safe level sl_n is randomly selected. From their safe levels, the safe level ratio is counted as:

$$sl_{ratio} = \frac{sl_x}{sl_n}.$$

Based on the values sl_x, sl_n, and sl_{ratio}, it is decided where the synthetic sample should be generated. The possibilities are:

- both instances represent a noise ($sl_x = 0$, $sl_n = 0$): no sample is generated
- n is a noise ($sl_n = 0$): x is duplicated
- $sl_x = sl_n, sl_{ratio} = 1$: new sample is generated along the line connecting x and n (same as for SMOTE 2.1)
- $sl_x > sl_n, sl_{ratio} > 1$: new sample is generated closer to x

- $sl_x < sl_n, sl_{ratio} < 1$: new sample is generated closer to n

The main idea is to generate new samples around safer levels, which are recognized by the number of minority and majority samples in the kNN. According to the authors, thanks to this, Safe-Level-SMOTE does not generate noisy samples.

2.1.3 Cluster-based Oversampling Methods

Cluster-based oversampling methods deal not only with the between-class imbalance but also with the within-class imbalance [21]. In the case of SMOTE, for example, the synthetic sample is generated between two minority class samples where one lies in the kNN of the other sample. However, this approach does not consider the majority samples that can appear between those minority samples. Therefore, if there are majority class samples between the neighboring minority class samples, the generated sample would be noise rather than a representative minority observation. Cluster-based algorithms deal with this problem by searching for clusters and oversampling only within those clusters. This approach helps to mitigate the potential generation of meaningless, noisy samples.

Most of the algorithms adopt k-means clustering algorithm. At first, k samples are randomly selected as centroids. In the second step, the distance of all samples to all centroids is calculated, and each sample is assigned the closest centroid. Then, the centroids are updated to be an average of all samples assigned to the same cluster. The second and third steps are repeated until the centroids and clusters change. The algorithm terminates when these two elements become stable. [22]

2.1.3.1 Cluster-Based Oversampling

Cluster-Based Oversampling (CBO) [22] was introduced in 2004, and it aims to solve the problems of both between-class and within-class imbalances. This algorithm first uses k-means clustering for each class separately to create clusters.

Then, the number of samples in each cluster is counted. For the majority class, each cluster is oversampled to get the same number of samples in each cluster as there are in the largest one. The minority class is oversampled in the way that each cluster contains the same number of samples after oversampling, and their total number is equal to the number of samples in the oversampled majority class. As the oversampling algorithm, ROS is used in CBO in the original article [22]. The authors claim that Cluster-based oversampling works well, especially for complex datasets with only a few samples.

2.1.3.2 Cluster-SMOTE

Cluster-SMOTE [23] was introduced in 2006 and is one of the simplest clustering-based oversampling algorithms. Its idea is to first apply clustering to the dataset, including only minority samples. For this purpose, k-means clustering is used in the original article. The second step is applying SMOTE to the dataset. However, the oversampling is done only within the created clusters of the minority samples. This leads to improvement of the oversampling on the localization basis of the new synthetic data. The described oversampling method idea can be seen in Figure 2.3.

Compared to CBO, both algorithms use k-means clustering. However, CBO does not concentrate only on the minority class but oversamples both classes and, therefore, deals with the within-class imbalance of both classes. Moreover, both techniques use different oversampling algorithms.

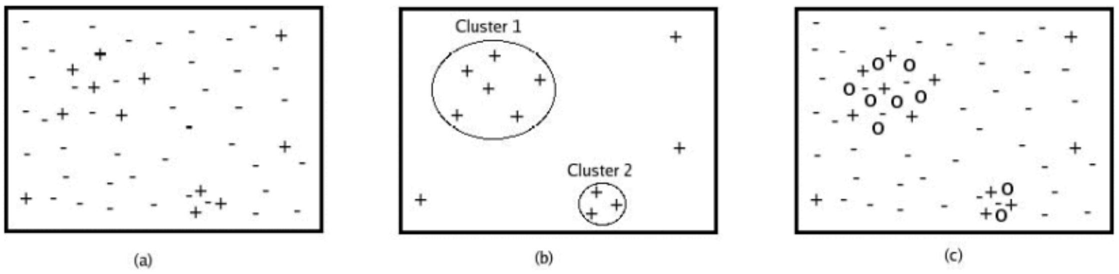


Figure 2.3 Cluster-SMOTE taken from [23]: (-) majority samples, (+) minority samples, (0) synthetic samples

2.1.3.3 MWMOTE

Majority Weighted Minority Oversampling TEchnique (MWMOTE) [24], as the name says, uses information from both classes for oversampling. The article presenting this method draws attention to the fact that methods such as Borderline-SMOTE and ADASYN, in some cases, fail to select correct samples for oversampling, or they generate samples into the majority class sample space. The goal of MWMOTE is to improve the selection of samples for oversampling and the generation itself.

The clustering of the minority class samples is used to ensure that the new samples are generated within the minority class clusters and not between majority class samples. The weight for selection for oversampling of each minority observation is assigned based on the sample importance. The importance is given by the distance from the decision boundary, the sparsity of the cluster the sample belongs to, and the sparsity of the closest majority class cluster.

These more complex steps for oversampling weight assigning ensure that correct probabilities are given to the minority class samples and that new samples are generated within minority class clusters.

2.1.3.4 k-means-SMOTE

k-means-SMOTE [25] is another oversampling technique that combines k-means clustering and SMOTE. This algorithm consists of three main steps: clustering, filtering, and oversampling.

In the first phase, the whole dataset is clustered regardless of the data labels using k-means clustering. In the filtering phase, only clusters having more than 50 % of their samples from the minority class are selected. Each of the selected clusters is assigned a sampling weight. The weight depends on the sparsity of the minority class samples within the cluster. Higher weight is given to the clusters with sparse minority areas. In the last oversampling phase, SMOTE is used to generate synthetic samples within the selected clusters (Eq. 2.1). The number of generated samples for each cluster depends on its sampling weight. The idea of k-means-SMOTE can be seen in Figure 2.4.

Compared to other methods, k-means-SMOTE aims to eliminate more false positives. It also detects safe areas for generating synthetic samples. This is done by clustering and filtering only those clusters having at least half of their samples from the minority class. This way, generating is not done for noisy samples as it is done in SMOTE. This algorithm deals with the within-class imbalance by assigning sampling weight based on the cluster sparsity.

2.1.4 Boosting Oversampling Methods

Those methods combine the advantages of traditional oversampling and boosting algorithms. Boosting is an ensemble method that sequentially uses weak learners, which together create a robust model. It focuses on the misclassified samples in each iteration of the ensemble learning.

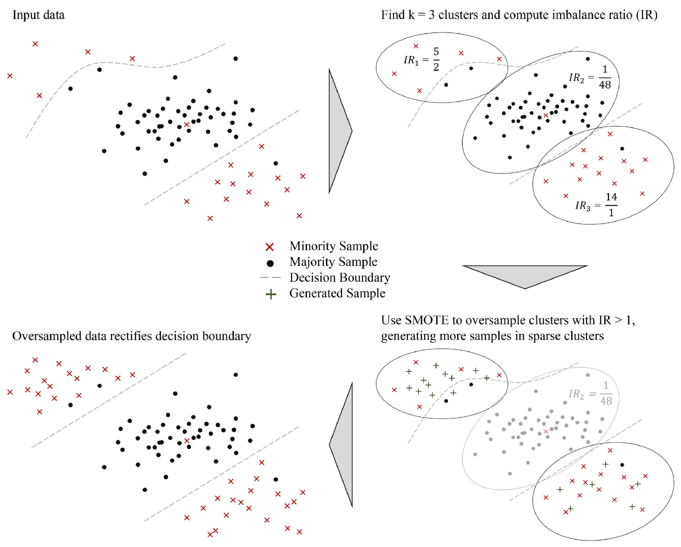

Input data
Find k = 3 clusters and compute imbalance ratio (IR)
$IR_1 = \frac{5}{2}$
$IR_2 = \frac{1}{48}$
$IR_3 = \frac{14}{1}$
× Minority Sample
● Majority Sample
Decision Boundary
+ Generated Sample
Oversampled data rectifies decision boundary
Use SMOTE to oversample clusters with IR > 1, generating more samples in sparse clusters

Figure 2.4 k-means-SMOTE taken from [25]

However, the weak learners still have to be trained on representative training data, and it can be problematic when having an imbalanced dataset. That is why ensemble methods are combined with oversampling techniques. Oversampling is integrated into each of the iterations of the boosting algorithm to create better predictions of the final model.

2.1.4.1 SMOTEBoost

SMOTEBoost [26] is an approach combining SMOTE and boosting, in the original article concretely AdaBoost.M2 [27]. In each boosting iteration, SMOTE is used to oversample the minority class. The idea of SMOTEBoost is to enlarge the training dataset in each boosting iteration by synthesizing the minority class, leading to a more balanced dataset. Each minority sample in each iteration is assigned a sampling weight. The weight is higher for misclassified samples and lower for correctly classified samples.

The advantage is that different samples are created and used for training in each iteration, which helps generalization. The synthetic samples are again removed from the training dataset after each iteration is done. This algorithm updates the data distribution by generating synthetic samples instead of traditional uniform updates. The predictions then can be done on the learned boosted classifier.

2.1.4.2 DataBoost-IM

DataBoost-IM [28], opposite to SMOTEBoost, uses both classes, does not concentrate on or favour any, and oversamples for both classes. In each boosting iteration, the algorithm finds hard-to-learn observations from each class separately. It uses those for generating synthetic samples, which are added to the training set, and the newly formed set is rebalanced. The weak learner is trained using this training set, and based on its performance, it is assigned a weight, which is used in the final prediction.

The hard-to-learn samples are selected so that all training samples are sorted based on their weights in descending order. Only defined top samples are used in the following steps. The amount depends on the classifier error and the size of the training set. Then, the numbers of majority and minority samples are selected based on their counts in the top samples.

In the data generation step, each class is taken separately again. Nominal and continuous values are also handled differently. For each nominal feature, the number of its values is counted, and in the same amount, they are randomly assigned to the synthetic samples. For each continuous attribute, the synthetic samples are generated from the same distribution as the actual data, considering the mean, standard deviation, and minimal and maximal values.

After generating the synthetic samples, each of them is assigned a weight, which is given by the fraction of the weight of the original sample it was generated from and the total number of samples generated from this original sample. The total weight for each class is computed, and based on their ratio, the weights of instances of the class with a smaller weight are multiplied by a fraction of the total weights so that the two weights are the same.

The main advantage of this method is concentrating on all hard-to-learn observations, generating samples for both classes separately, and reweighting accordingly to have a balanced dataset.

2.1.4.3 RAMOBoost

Ranked Minority Oversampling in Boosting (RAMOBoost) [29] is a method that combines adaptive synthetic data generation with boosting. Same as in ADASYN, each minority sample is assigned a weight based on the number of majority class samples within its kNN. The difference between those two algorithms is that ADASYN directly assigns each sample number of synthetic samples to be generated from it, and samples with no majority samples in their kNN are not considered for oversampling. On the other hand, RAMOBoost only assigns each minority sample a probability of being chosen for oversampling. Hard-to-learn samples near the decision boundary have a higher probability of being chosen, but it is not absolute as in the case of ADASYN.

RAMOBoost uses the AdaBoost.M2 algorithm for boosting. Each iteration of the algorithm corresponds to one weak learner. In each iteration, mislabeled data are sampled and given a probability for oversampling. The boosting procedure is similar to SMOTEBoost, but because of the adaptive weighting, RAMOBoost performs better, based on the RAMOBoost article results.

2.1.5 Relabeling Methods

Relabeling methods do not add or remove new samples to or from the dataset, as it is done when oversampling or undersampling is used. Still, they balance the dataset by changing the class label of some observations.

2.1.5.1 SPY

For SPY [21], the majority class samples lying close to the decision boundary are marked as SPY samples, and their label is changed to the minority class label. This way, the number of majority class samples decreases, and the number of minority class samples increases. This approach is particularly interesting because in the work by Kovács [6], this method was discovered to be the fastest of all 85 methods used in the experiments, and therefore, it is presented in this thesis. Change of the decision boundary when SPY is used on a dataset can be seen in Figure 2.5.

2.2 Generative-based Algorithms

In this Section, generative-based oversampling methods are presented. Based on their architecture and generation procedure, those techniques are divided into GAN-based, AE-based, GAN+AE-based, and Score-based.

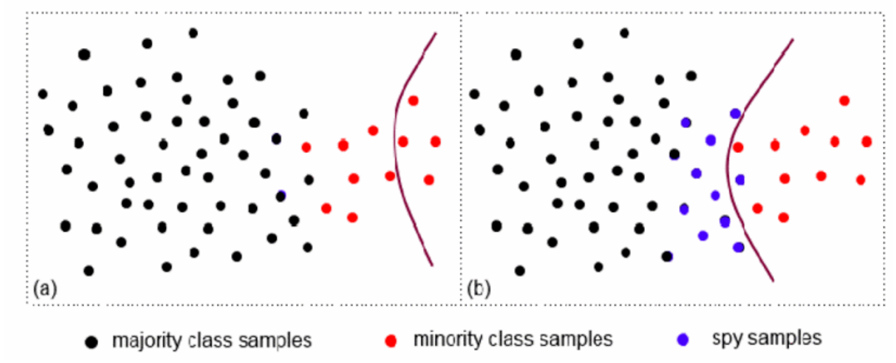


Figure 2.5 SPY taken from [21]: Relabeling

2.2.1 GAN-based

In recent years, generative models have become popular because of their excellent performance, especially on image datasets. GANs were first introduced in 2014 by Ian J. Goodfellow [12], and since then, they have been used extensively. GAN consists of two neural networks: a generative model (generator G) and a discriminative model (discriminator D). The generator produces fake samples from the latent space and tries to fool the discriminator. The discriminator is fed with real and fake observations generated by the generator, and its task is to distinguish fake samples from real ones. G and D play the so-called minmax game, the mathematical expression of which looks as follows:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))] \tag{2.2}$$

where $p_z(z)$ represents an input noise, p_{data} is the real data distribution, and $D(x)$ represents a probability that x is from real data [12].

After training both G and D, the generator is used for synthetic data generation. GANs are popular mainly for generating images but can also be used for tabular data generation. However, not many GAN architectures are currently focusing on oversampling the minority class exclusively for datasets with a mix of feature types.

2.2.1.1 TableGAN

TableGAN [30] is a generative model that concentrates primarily on the privacy problem. Its primary purpose is to generate synthetic data that can be used instead of real observations as training data. The reason for this is that the real data can contain sensitive information that should not be leaked. TableGAN, therefore, generates synthetic data that should minimize the possibility of information leakage while trying to maximize model compatibility. The trade-off between those two requirements is controlled by the parameters of the model. Model compatibility means that the performance of a machine learning model learned on real and synthetic datasets is similar regarding the test set. This algorithm is not specialized in the IDS problem but can be used for synthesizing data of the minority class as well.

TableGAN is a GAN-based model having three main components: generator, discriminator, and classifier. The roles of the generator and discriminator are the same as in vanilla GAN [12]. However, the classifier is added parallel to the discriminator to ensure the semantic integrity of the synthetic samples. This means that the generator is penalized for generating samples that do not make sense (a combination of their feature values is not seen in the real observations).

TableGAN also changes its loss function compared to vanilla GAN. There are three types of loss functions: original loss from deep convolutional GAN (DCGAN)[31], information loss, and classification loss. The information loss ensures preserving the same statistical aspects as the real data. The classification loss ensures semantic integrity.

The architecture of TableGAN is based on DCGAN. Each component (generator, discriminator, classifier) is represented by a convolutional neural network (CNN); hence, the data have to be converted into a square matrix and padded if needed. The architecture of the classifier is the same as the discriminator's, differing only in the output layer. The network is trained on real observations and learns the correlation between features to ensure that the generator produces semantically correct data.

TableGAN is, based on the original paper result, the best method for handling the privacy preservation and model compatibility trade-off. When dealing with the IDS problem, the parameter controlling the trade-off can be adjusted to create more compatible data with the original dataset. In the work by Kim et al. [32], TableGAN was used for oversampling and, as it is a simpler method than further presented CTGAN, it gave worse results.

2.2.1.2 TGAN

Tabular GAN (TGAN) [33] is a model similar to TableGAN. However, there are differences between those two algorithms: TGAN generates data feature by feature, concentrating on each one with respect to the previous ones, whereas TableGAN focuses on the whole data sample at once.

TGAN introduces mode-specific normalization for continuous features. The distribution of continuous features can be multimodal. Therefore, normalization into $[-1, 1]$, $[0, 1]$, or standardization would not work well. The mode-specific normalization idea is to assign each sample a mode and normalize data within each mode. The data are clustered into the modes using the Gaussian Mixture Model (GMM) with a fixed number of components. The categorical variables, on the contrary, are handled by using smoothing: one-hot encoding is used, and noise is added to each binary value. Finally, the values are normalized.

The architecture of TGAN consists of long-short-term memory (LSTM) with an attention mechanism for the generator and multi-layer perceptron (MLP) for the discriminator. The architecture can be seen in the Figure 2.6.

TGAN is a method used for synthetic tabular data generation. It can generate samples of both classes to replace real data, or it can be used for oversampling, as shown in the work by Quintana et al. [34]. The advantage of this algorithm is straightforward post-processing, allowing data reconstruction with the original structure.

2.2.1.3 CTGAN

TGAN was introduced in 2018, and a year later, the same authors presented another generative method that builds upon TGAN, namely Conditional Tabular GAN (CTGAN) [35]. As the name says, CTGAN conditions its generator while TGAN generates data in an unconditioned manner. The model can be conditioned by the categorical columns, and this architecture addresses the problem of having imbalanced categorical features. CTGAN uses mode-specific normalization, as TGAN does. However, CTGAN applies a variational Gaussian mixture model (VGM) instead of GMM, which means that the number of modes is computed and is not fixed as in the case of GMM. What is more, a conditional generator and training-by-sampling are implemented in the CTGAN architecture.

CTGAN concentrates not on the imbalance of the target feature but on the imbalance within each categorical column. All categories in a categorical column should be sampled evenly. The generator loss is modified by adding a cross-entropy loss to penalize the generation of data that differ from the conditional vector.

Training-by-sampling is used in CTGAN, which refers to sampling the conditional vector properly. The feature for conditioning is selected randomly, with all features having the same

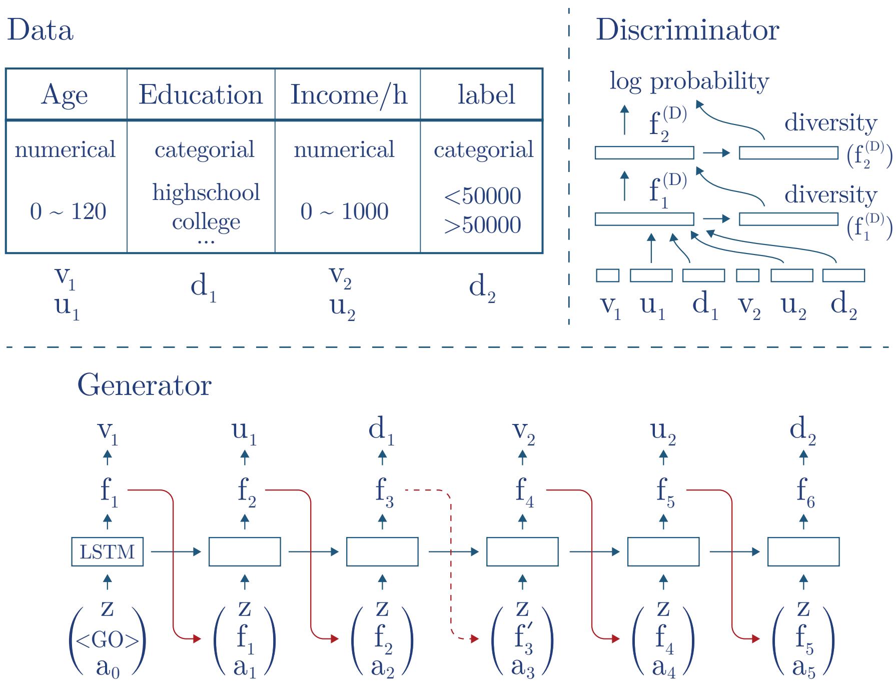


Figure 2.6 TGAN taken from [33]

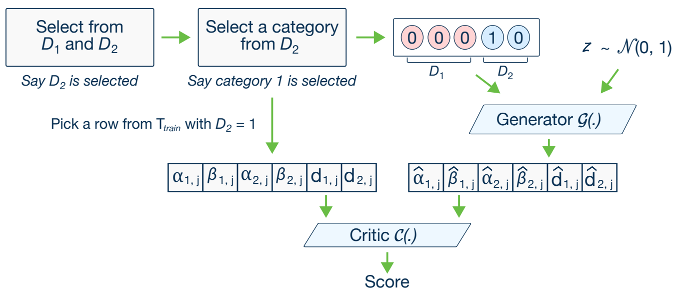

Select from D1 and D2
Say D2 is selected
Select a category from D2
Say category 1 is selected
0 0 0 1 0
D1
D2
z ~ N(0, 1)
Pick a row from T_train with D2 = 1
Generator G(.)
α1,j β1,j α2,j β2,j d1,j d2,j
α̂1,j β̂1,j α̂2,j β̂2,j d̂1,j d̂2,j
Critic C(.)
Score

Figure 2.7 CTGAN taken from [35]

probability. However, the category within the feature is selected randomly according to the probability mass that corresponds to the logarithm of the frequency of each category. This ensures proper exploration of the feature space, giving minority-represented categories a chance.

When CTGAN is used for oversampling, the target feature is marked as a categorical feature to ensure its proper sampling. In the work by Eom et al. [36], CTGAN is used for oversampling a medical dataset of Parkinson's disease dementia patients. How CTGAN works can be seen in Figure 2.7.

Another difference from TGAN is in the architecture. In CTGAN, both the generator and discriminator consist of fully connected layers instead of LSTMs.

The advantage of CTGAN is more controlled training and sampling because of the usage of a conditional generator. The results also show that CTGAN is more resistant to mode collapse than other methods, from the presented ones concretely MedGAN and TableGAN. Mode collapse is a scenario when the generator learns to produce samples that fool the discriminator, but the generator's output lacks variety as it maps each input into those indistinguishable synthetic samples instead of producing diverse output.

2.2.1.4 cWGAN

Conditional Wasserstein GAN (cWGAN) [37] is a method that can generate data having categorical and numerical features and concentrates on the IDS problem exclusively. cWGAN combines conditional GAN (cGAN) [38] and Wasserstein GAN (WGAN) [39].

The numerical features are scaled into $[0, 1]$, and Gaussian noise is added to the normalized feature value of both real and synthetic observations. The categorical features are one-hot encoded, and *Gumbel-softmax* [40] activation function is used on their output. *Gumbel-softmax* is computed as follows:

$$\text{Gumbel-softmax}(x_i) = \frac{\exp((x_i + g_i)/\tau)}{\sum_{j=1}^{k} \exp((x_j + g_j)/\tau)} \qquad \text{for} \qquad i = 1, ..., k \tag{2.3}$$

where $g_1, ..., g_k$ are drawn i.i.d. from $Gumbel(0, 1)$, x_i is one component of vector x, and τ is an adjustable temperature parameter, which controls the smoothing. This results in soft one-hot encoding that is differentiable [37].

Having tabular data, the features might correlate. Therefore, the numerical output of the generator is conditioned by the categorical output, and the authors call this approach self-conditioning.

The conditioning in cWGAN is not done as in CTGAN, where the conditional vector is used, but cWGAN uses conditioning on the target feature, concretely minority class label, during the sampling. As the name implies, Wasserstein loss with gradient penalty [41] is used in cWGAN. The objective function of Wasserstein GAN with gradient penalty is:

$$\min_G \max_D {}_{x \sim p_{\text{data}}}[D(x)] - {}_{z \sim p_z}[D(G(z))] - \lambda_{\hat{x} \sim p_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2] \tag{2.4}$$

where λ is the gradient penalty coefficient and $\hat{x}$ represents the linear interpolations between real and fake samples [37].

Moreover, in cWGAN, an auxiliary classifier (AC) with architecture similar to the discriminator's is used parallel to the discriminator to ensure the generation of samples belonging to the class used as a condition. It is done by adding the AC loss into the cWGAN objective function:

$$\lambda_{AC\, z \sim p_z}[BCE(AC(G(z)))]$$

where BCE is the binary cross entropy between the predicted class label probability and the true class label.

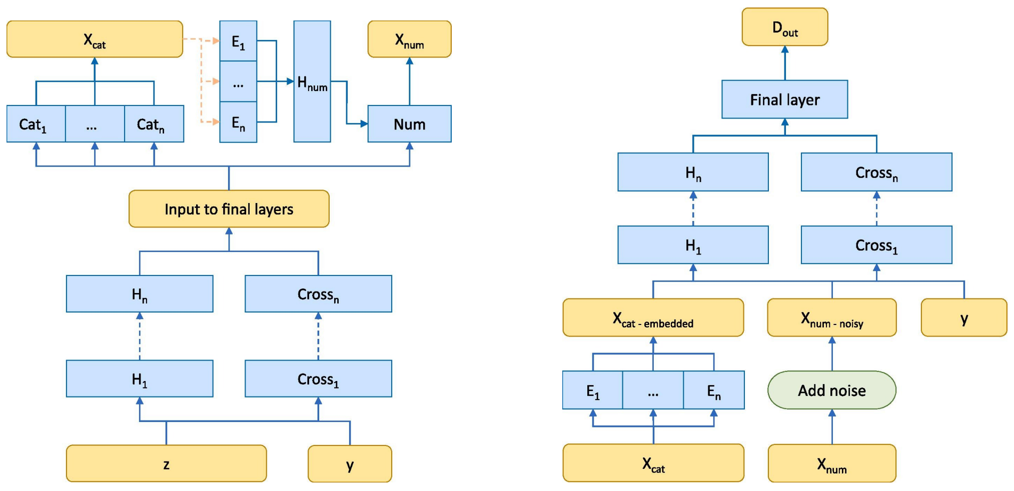
cWGAN generator and discriminator can be seen in Figure 2.8.

In the original article [37], the authors claim that synthetic data generation for re-balancing datasets using cWGAN results in better performance of the classifiers in the downstream task than re-balancing using traditional methods such as SMOTE and ADASYN.

2.2.1.5 CTAB-GAN

Conditional Table GAN (CTAB-GAN) [42] is one of the state-of-the-art generative methods for tabular datasets. It was introduced in 2021. CTAB-GAN addresses privacy problems, strict regulations (GDPR) as well as data imbalance and skewed distribution problems. Compared to other methods, CTAB-GAN can handle not only continuous and categorical features but also a combination of continuous features with special discrete values. The authors call this a mixed variable or mixed data type. Other methods handle those features as continuous, which is not correct due to the wrong estimation of fixed numbers with meaning (e.g. number zero). In the article [42], a Mixed-type Encoder is presented to deal with mixed variables.

CTAB-GAN, compared to CTGAN, adds classification loss and encodes all features, regardless of the data type, into the conditional vector. The generator can produce semantically incorrect data. Therefore, CTAB-GAN, the same as TableGAN, adds a classifier with classification



(a) generator **(b)** Discriminator

Figure 2.8 cWGAN taken from [37]: Generator and Discriminator

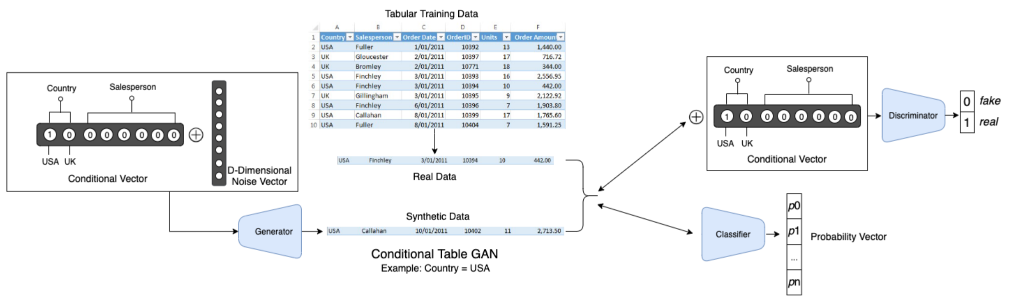


Figure 2.9 CTAB-GAN taken from [42]: Synthetic Data Generation

loss to the architecture to ensure semantic integrity. CTAB-GAN is inspired by other methods and combines their advantages, e.g., it uses information loss as TableGAN does, has a conditional generator with generator loss, and uses training-by-sampling as CTGAN does. What is more, a log-frequency sampler is used to address the problem of mode collapse.

Training-by-sampling in the case of CTAB-GAN does not include only categorical features in the conditional vector but rather all features: continuous, categorical, and mixed. The log probability, instead of the frequency of each mode or category, is used when generating the conditional vector. Log probability distributes the chance of being selected more evenly, giving a higher chance to the minority. This also helps to alleviate the IDS problem for continuous and mixed variables. The correlation of all features is also captured by the generator when all of them occur in the conditional vector. A logarithmic transformation is also used for preprocessing the continuous features having a long tail before their encoding because it has been observed that the encoding of such variables is difficult.

The architecture of CTAB-GAN consists of three neural networks: generator, discriminator, and classifier. The generator and discriminator networks are the same as those of TableGAN, using CNNs. The classifier is implemented as MLP. CTAB-GAN synthetic data generation process can be seen in Figure 2.9.

The results in the work by Zhao et al. [42] show that CTAB-GAN outperforms other methods like TableGAN, MedGAN, and CTGAN based on the machine learning (ML) utility, meaning that the classifiers perform the most similarly when it is trained on the real training dataset and synthetic dataset and further tested on the real test dataset. The benefits of this method are mainly the construction of the conditional vector, including all feature types and the addition of a classifier into the architecture to ensure semantic integrity.

2.2.1.6 TTGAN

Tabular Translation Generative Adversarial Network (TTGAN) [43] is a technique for minority class oversampling on binary classification tabular datasets. The idea is to use majority class samples as input to the generator and translate them into synthetic minority class samples. What is more, by adjusting the loss function, the generated data are placed close to the decision boundary.

The network architecture has four parts: 2 generators, G and G', and 2 discriminators, D and D'. G and D are the generator and discriminator from the traditional GAN architecture for adversarial training. G' and D' are networks used for sampling from the majority class in CycleGAN [44], upon which TTGAN builds. All parts of TTGAN are multi-layer, fully connected networks.

Next to the adversarial loss (Eq. 2.2), TTGAN intercorporates three other loss functions: translation loss and cycle-consistency loss and identity loss from CycleGAN. Translation loss $\mathcal{L}_T$ is used to minimize the distance between the generator's input sample and its output translated sample:

$$\mathcal{L}_T(z) = ||z - G(z)||_1.$$

Cycle-consistency loss $\mathcal{L}_C$ ensures that translating back and forth should result in getting a sample close to the initial one:

$$\mathcal{L}_C(z) = ||G(G'(x_+)) - x_+||_1 + ||G'(G(x_-)) - x_-||_1.$$

Identity loss $\mathcal{L}_I$ assures that when a sample of the output domain is used as a generator's input, the mapping should be close to the identity mapping:

$$\mathcal{L}_I(z) = ||G(x_+) - x_+||_1 + ||G'(x_-) - x_-||_1$$

where x_+, x_- represent minority, resp. majority class samples, and z is sampled from the majority class samples in the training dataset.

The generator's objective function is:

$$\mathcal{L}^G = \mathcal{L}^G_{orig} + \lambda_T \mathcal{L}_T + \lambda_C \mathcal{L}_C + \lambda_I \mathcal{L}_I$$

where $\lambda_T, \lambda_C, \lambda_I$ are hyperparameters of the model. The discriminator's objective function stays the same as in the original GAN objective function: $\mathcal{L}^D = \mathcal{L}^D_{orig}$.

What is more, not all of the generated samples are added to the original training dataset. They are sorted in descending order based on their likelihood of belonging to the minority class. Based on the cut-off limit, samples with scores that are too high are removed because they represent data far from the decision boundary. From the rest of the samples, only a specified number, which is a multiple of the minority class samples, is retained and added to the original training dataset.

The generator does not generate data from noise but rather translates from the majority class: $z \sim X_-$. This is the main difference from the above-presented GAN-based techniques.

2.2.2 AE-based

Autoencoders (AE) represent a bidirectional mapping scheme [45]. AE architecture consists of two neural networks, namely, encoder and decoder. The encoder's function is to map input data to lower dimensional latent space, whereas the decoder maps data from the latent space to the original space. The goal is for the decoder to reconstruct the input by minimizing the reconstruction error.

2.2.2.1 TVAE

TVAE was introduced together with CTGAN in the work by Xu et al. [35]. It is a generative model that can be used for data generation as well as for dataset oversampling of a particular class. It is implemented as a variational autoencoder (VAE) for tabular data generation and uses the same preprocessing as CTGAN. TVAE consists of two neural networks: an encoder and a decoder. The architecture of both consists of fully connected layers. In the original paper, the performance of TVAE is competitive with CTGAN.

2.2.2.2 MGVAE

Majority-Guided Variational Autoencoder (MGVAE) [46] was introduced in 2023, and it focuses exclusively on the IDS problem. It is an approach that "generates new minority samples under the guidance of a majority-based prior" [46]. This algorithm can effectively deal with the problem of having a big IR and only a few minority-class samples. The idea is that the classes are related, and the generation of samples from the minority class can benefit from the knowledge of the majority class. "MGVAE generates the minority sample according to a majority-based prior, resulting in one-to-one sample mapping." [46]

MGVAE architecture consists of fully connected layers for tabular datasets. However, in the case of bigger image datasets, the architecture uses CNN. This shows the adaptability of the method and its usage in different domains.

The training of the model has two parts: pre-training on the majority class samples and fine-tuning on the minority class samples. This is done to avoid mode collapse and overfitting. The fine-tuning has to be controlled in order to prevent forgetting the training on the majority class. This is done by adding the Elastic Weight Consolidation [47] regularization that penalizes the model parameter changes.

The sampling has three parts: firstly, data are drawn randomly from the majority-based prior. Secondly, latent codes are sampled in the latent space conditioned by the data selected in the first phase. In the last part, those sampled latent codes are transformed via a decoder to create minority-class synthetic samples.

This method can be used on images as well as on tabular datasets. Its advantage is that it uses the information from both classes, minority and majority, to generate diverse synthetic samples whose usage helps to avoid overfitting in the downstream tasks.

2.2.3 GAN+AE-based

As different oversampling architectures of GANs and AEs exist, there are also models that use a combination of those two.

2.2.3.1 MedGAN

Medical Generative Adversarial Network (MedGAN) [48] is a state-of-the-art model for generating realistic data similar to electronic health records of patients. MedGAN combines an autoencoder with GAN and is meant for generating high-dimensional data with discrete variables. In the original article [48], minibatch averaging is introduced, which is a technique used to deal with the mode collapse problem.

The MedGAN architecture contains four parts: generator and discriminator of GAN and encoder and decoder of VAE. The architecture is shown in Figure 2.10. The trained decoder

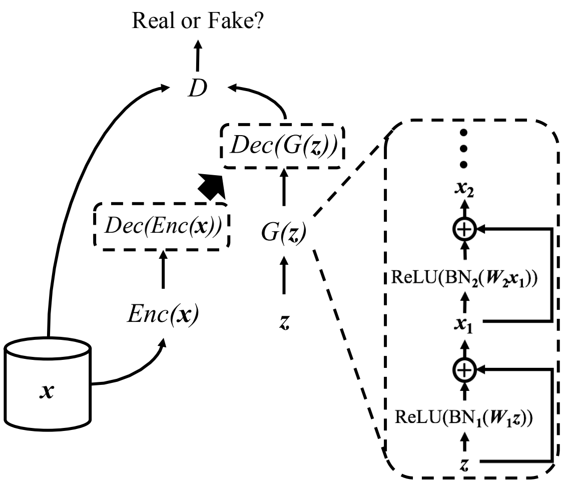


Figure 2.10 MedGAN taken from [48]

is used on the continuous output of the generator to create the discrete output before it is fed into the discriminator.

First, VAE is trained to minimize the reconstruction error between the input and output data. Secondly, GAN is trained. However, the generator does not generate realistic data directly. It generates a representation that serves as an input to the pre-trained decoder, which generates synthetic patient data based on its input.

The authors of MedGAN also introduced minibatch averaging to deal with mode collapse. Minibatch averaging allows the discriminator to view the whole minibatch of input samples as an average, and the objective function is modified to work with the average accordingly.

The results of MedGAN were also shown to a doctor who could distinguish the real and fake data only if there was a semantic mistake. This shows the possibility of using MedGAN in healthcare and generating realistic synthetic tabular samples. Although the MedGAN algorithm does not concentrate exclusively on the IDS problem, it can be used for minority data oversampling, as it is done in other articles focusing on this problem [32].

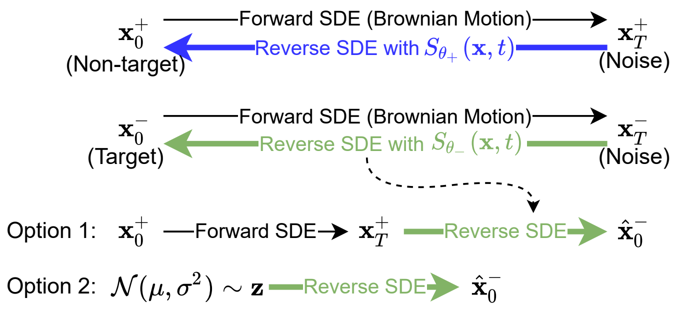
2.2.4 Score-based

Score-based generative models (SGMs) represent another type of generative model next to GANs and AEs. They are primarily used for generating images. SGMs use the stochastic differential equation (SDE). They are similar to other generative methods from the architectural point of view. Like GANs or AEs, their network consists of two parts: one for the forward diffusion process and the reverse one for the de-noising process. During the diffusion process, noise is added to the data. In contrast, during the de-noising process, as the name suggests, it is removed in order to reconstruct the original data as best as possible. [32]

2.2.4.1 SOS

Score-based OverSampling (SOS) [32] is, according to the authors, the first score-based generative oversampling method, and therefore, the score neural network had to be re-designed accordingly for the purpose of generating tabular data instead of images. SOS can handle both continuous and categorical features. It uses one-hot encoding for categorical features and normalization into the interval $[0, 1]$ for continuous features.

The score network consists of fully connected layers. The idea of SOS is to train SGMs for



$\mathbf{x}_0^+$
(Non-target)
Forward SDE (Brownian Motion)
Reverse SDE with $S_{\theta_+}(\mathbf{x},t)$
$\mathbf{x}_T^+$
(Noise)
$\mathbf{x}_0^-$
(Target)
Forward SDE (Brownian Motion)
Reverse SDE with $S_{\theta_-}(\mathbf{x},t)$
$\mathbf{x}_T^-$
(Noise)
Option 1: $\mathbf{x}_0^+$ Forward SDE $\mathbf{x}_T^+$ Reverse SDE $\hat{\mathbf{x}}_0^-$
Option 2: $\mathcal{N}(\mu,\sigma^2) \sim \mathbf{z}$ Reverse SDE $\hat{\mathbf{x}}_0^-$

Figure 2.11 SOS Concept taken from [32]: The authors use different notation and denote majority class samples with + and minority class samples with −.

each class separately. Then, the forward SDE for the majority class is used to create a noisy sample x_T^-, and the reverse SDE for the minority class is used on x_T^- to create a synthetic minority sample $\hat{x}_0^+$. The authors compare this approach to Borderline-SMOTE, as the samples are also generated near the decision boundary. Moreover, Kim et al. [32] create the synthetic samples without using the forward SDE for the majority class and sample from noise, too. The authors also compare this approach to SMOTE, as the synthetic samples are not created informatively, only near the decision boundary. Both approaches can be seen in Figure 2.11.

In the original article [32], the results show that SOS surpasses other oversampling methods, both traditional and generative-based, such as CTGAN, TableGAN, MedGAN, SMOTE, and Borderline-SMOTE. Thus, score-based generative methods look as promising ways to solve the imbalance dataset problem.

2.3 Combined Algorithms

This Section presents oversampling techniques that combine traditional methods such as SMOTE and generative algorithms, both GANs and AEs.

2.3.1 TAEI

Tabular AutoEncoder Interpolation (TAEI) [45] is a method combining AE and an interpolation method. TAEI deals with the problem of high-dimensional data. It maps them to a lower-dimensional dense latent space. Synthetic samples are then interpolated in this space. Thus, the oversampler does not have to deal with the problem of having categorical and continuous data types or sparse distribution, but those problems are handled by the autoencoder.

The architecture contains three parts: an encoder and decoder of AE and an oversampler. The algorithm first uses the encoder that maps original data to a continuous latent space. There, the oversampler creates synthetic data by interpolation, and the decoder maps those data back to the feature space. In the original article [45], more AE schemes are used, concretely: AE, VAE, Regularized AE, Adversarial AE, Interpolate Adversarial AE, and Adversarially Constrained AE Interpolation. Poly and SMOTE are used as oversamplers. However, the algorithm can also be implemented with other interpolation methods. The scheme of TAEI can be seen in Figure 2.12.

In the work by Darabi et al. [45], the authors optimize two metrics: Cover and Error. Cover shows how well synthetic data represent the real data, and Error represents the synthetic data reliability. Ideally, the balance between those metrics should be found. The results show that the performance of TAEI is higher, considering those two metrics, than the performance of using only the oversampler applied to the original data.

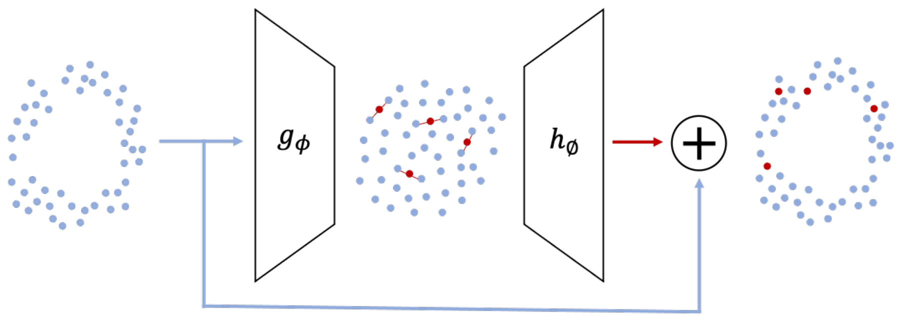
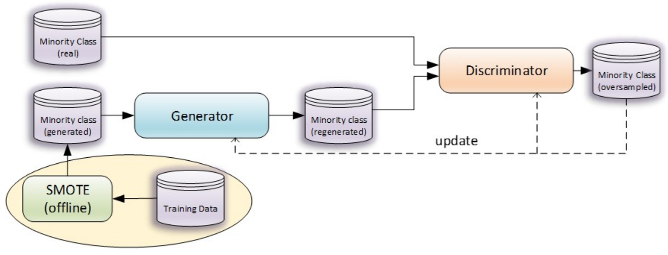


Figure 2.12 TAEI taken from [45]



Minority Class (real)
Minority class (generated)
Generator
Minority class (regenerated)
Discriminator
Minority Class (oversampled)
update
SMOTE (offline)
Training Data

Figure 2.13 SMOTified-GAN taken from [49]

2.3.2 SMOTified-GAN

SMOTified-GAN [49] is a method combining SMOTE for minority class oversampling and GAN for data generation. The idea is to first run SMOTE to create synthetic minority class samples. The second step is training GAN on those synthetic data and real minority class data. The generator takes the synthetic data as input and adjusts them to be similar to the real ones so that the discriminator is unable to distinguish between the fake and real minority data.

The combination of SMOTE and GAN should overcome the weaknesses of SMOTE, as it can create samples that do not match the original data distribution. Therefore, GAN is used to adjust those samples to create better ones that match this distribution. The generator does not sample from noise but is fed with data generated by SMOTE. The model of SMOTified-GAN can be found in Figure 2.13.

2.4 Summary

Comparing the traditional, generative-based, and combined oversampling techniques, the first mentioned do not need training as synthetic samples are generated immediately based on the given training data. On the other hand, generative-based and combined methods require training, which can be computationally demanding, but the sampling itself is then simple and fast.

Figure 2.14 shows the timeline of the origin years of all presented methods. There, it can be seen that traditional oversampling methods were mainly introduced between years 2002 and 2015, while generative-based and combined methods have gained popularity in the last six years. Dozens of traditional oversampling methods exist, including state-of-the-art methods like Borderline-SMOTE or ADASYN. On the contrary, the field of oversampling with generative-based methods is still not much researched, and this can be expected to change in the future.

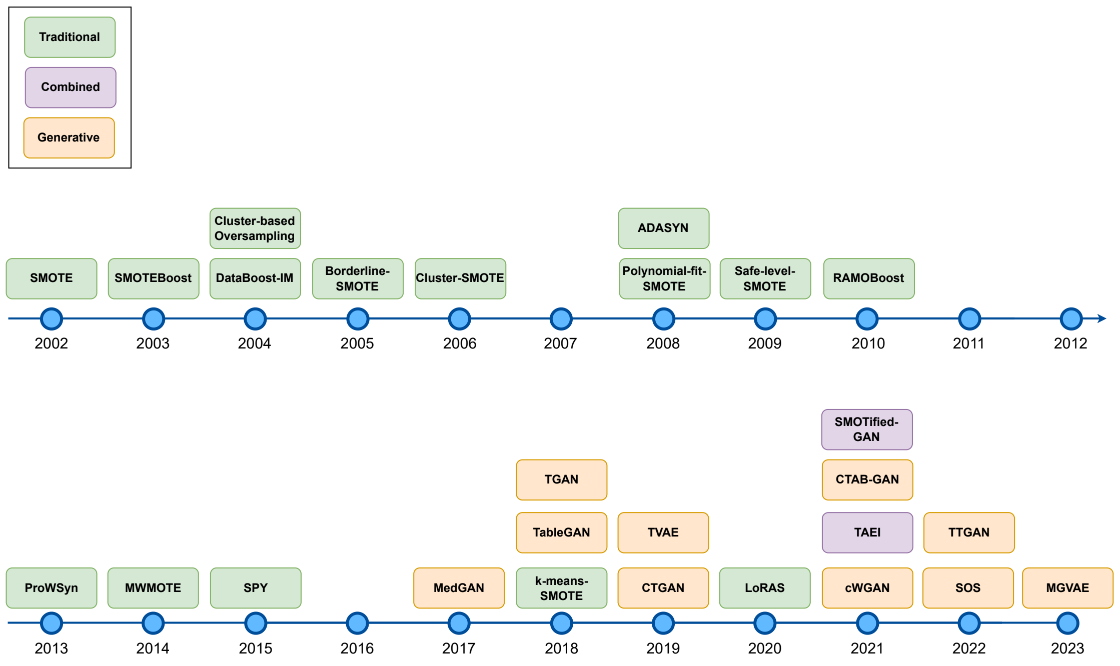


Figure 2.14 Timeline of Presented Methods Origins

Chapter 3

Selected Methods

In this Chapter, methods selected for implementation, evaluation, and comparison are presented in more detail. In the practical part of this thesis, for balancing the training dataset, we want to compare all types of methods: traditional, generative-based, and combined. For the experiments, methods with publicly available implementation are used.

From the traditional oversampling techniques for the implementation, SMOTE, Polynom-Fit-SMOTE, LoRAS, Borderline-SMOTE, and k-means-SMOTE were selected. From the generative-based techniques, both GAN-based and VAE-based are represented concretely by CTGAN and CTAB-GAN for the first type and by TVAE for the latter type. The combined techniques are represented by TAEI and SMOTified-GAN.

3.1 SMOTE

SMOTE [14] is one of the oldest oversampling algorithms that is widely used also nowadays, and it serves as a baseline for comparison with other methods. Its implementation used in this thesis comes from `imblearn` library[1]. SMOTE is defined in Section 2.1.1.1, where the equation for a synthetic sample generation is also presented (Eq. 2.1).

The synthetic sample is placed on the line connecting the selected minority sample x and its neighbor $x_{neighbor}$ that is selected from the k nearest neighbors. k is specified by SMOTE parameter `k_neighbors`, and its default value is 5. Another of the SMOTE `imblearn` implementation's parameters is the `sampling_strategy` that defines which classes should be oversampled. In our case, `minority` is chosen to oversample only the minority class to get a balanced dataset.

3.2 Polynom-Fit-SMOTE

Poly [10] was selected from the traditional methods. In the work by Kovács [6], it is the overall best traditional oversampling technique from 85 techniques used. There exists a library called `smote-variants` [50] that serves as an implementation for the article comparing all 85 algorithms. A version of Poly from this library is used in our experiments, concretely we chose `polynom_fit_SMOTE_mesh` as Mesh and Star topologies give the best results in the original article [10].

This algorithm takes as a parameter `proportion`. The number of generated minority samples G is computed as:

[1] `https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html`

$$G = proportion \cdot (N_- - N_+).$$

When `proportion` is set to 1, the resulting dataset will be balanced.

The idea behind Poly is: “Curve Fitting Methods are used to find a coefficient of a polynomial *p(x)* of degree n that fits the minority instances.” [10] Considering Mesh topology: for each minority sample in the feature space, a straight line is added, connecting this point with another minority class sample point. This is done for all minority class observations. Each line can be described as:

$$f_i(x) = ax + b.$$

The coefficients a and b have to be defined to fit the training minority data. Based on the `proportion`, k linearly-spaced value x_k is generated, where $k \in [-1, 1]$. The synthetic sample is generated by evaluation $f_i(x)$ at x_k.

3.3 LoRAS

LoRAS [18] was added to our implementation because it is one of the newest traditional methods with publishing year 2020. There exists a publicly available implementation on GitHub[2] which is used in this thesis.

“LoRAS oversamples from an approximated data manifold in the minority class.” [18] Having `|F|` features, the authors of LoRAS define *small dataset* as a dataset where for the number of samples n and the number of features `|F|` holds: $log_{10}(\frac{n}{|F|}) < 1$. When a sample's k neighborhood is selected, it creates such a small dataset. Let us call those selected minority neighboring samples as parents. For each parent, noise from a normal distribution $\mathcal{N}(0, h(\sigma_f))$, where $h(\sigma_f)$ is some function of the sample variance σ_f for feature $f \in F$, is added to each feature f from `F`. This way, m shadowsamples can be created, where $k \cdot m >> |F|$.

$|F|$ shadowsamples are then chosen by randomly selecting a parent, and shadowsamples are generated from this parent. Random affine combination of the selected $|F|$ samples creates one Localized Random Affine Shadowsample (LoRAS) in the same $|F| - 1$-dimensional plane. This plane is assumed to be “the approximation of the latent data manifold in the small neighborhood ” [18] that locally approximates the whole data manifold. Thus, the generated samples are assumed to be representative. Illustrative picture of LoRAS creation can be seen in Figure 2.2.

There is one callable method for oversampling with LoRAS, concretely `fit_resample`, which takes the minority samples and the majority samples of the training dataset as parameters. Other parameters, such as the number of shadow points or the number of generated points, are optional and computed within the function when not specified.

3.4 Borderline-SMOTE

Borderline-SMOTE [13] represents the traditional borderline oversampling methods outlined in Section 2.1.2. Same as SMOTE, its implementation is part of the `imblearn` library[3]. As mentioned in Section 2.1.2.1, there are two Borderline-SMOTE variants: Borderline-SMOTE1 and Borderline-SMOTE2. The second version of the algorithm is used in the experiments of this thesis.

Having N_+ minority and N_- majority samples in the training dataset, for each minority sample x, its *k1* nearest neighbors are calculated from all training samples. Let us denote the number of majority samples within the neighbors as m. There are three scenarios for each x:

[2]`https://github.com/sbi-rostock/LoRAS`

[3]`https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.BorderlineSMOTE.html`

1. $k1 = m$: x is considered to be a noise
2. $\frac{k1}{2} \leq m < k1$: x is considered to be a borderline sample and is added to `DANGER` set
3. $0 \leq m < \frac{k1}{2}$: x is considered to be a safe sample

For each sample x in `DANGER`, its $k2$ nearest neighbors are computed, and some of those neighbors are randomly selected for synthetic data generation. The synthetic samples are generated as in SMOTE (Eq. 2.1) when the neighbor is a minority class observation. When the neighbor is a majority sample, the synthetic sample is generated closer to the sample x. This means that the interval for δ in Eq. 2.1 is $\delta \in [0, \frac{1}{2}]$.

Same as SMOTE, the `imblearn` implementation takes the `sampling_strategy` as a parameter. Other parameters are `k_neighbors` and `m_neighbors` denoting the number of kNN used in the algorithm, above denoted as $k1$ and $k2$. Their default values are 5, resp. 10.

3.5 k-means-SMOTE

k-means-SMOTE [25] represents, in the experimental part of this thesis, the cluster-based traditional oversampling techniques described in Section 2.1.3. Its implementation is also available in the `imblearn` library[4]. Coming from the same library as SMOTE and Borderline-SMOTE, k-means-SMOTE also has the `sampling_strategy` parameter and it is set to `minority` to receive balanced dataset where only minority class is oversampled. A parameter `n_cluster` denoting the number of clusters can be specified, and its default value is 8. During the implementation, we had trouble selecting the proper number of clusters for some datasets. Even a big number of clusters with lowering the `cluster_balance_threshold` resulted in error. This is the reason why the results are not presented in Chapter 6 for the combination of k-means-SMOTE and those problematic datasets.

This algorithm concentrates on cluster density and generates samples in sparse areas. This way, not only the between-class imbalance but also the within-class imbalance of the minority class is handled. It is achieved in three steps: clustering, filtering, and oversampling. The clustering is done regardless of the class label using the k-means clustering algorithm described in Section 2.1.3. In the second phase, only those clusters having a high number of minority samples are retained. This is where the parameter `cluster_balance_threshold` plays its role. The synthetic data generation is then executed as in SMOTE (Eq. 2.1) with the difference that oversampling is done only within the filtered clusters, having more samples generated in sparse areas of the minority class.

The sparsity of each selected cluster c has to be computed. This is done in more steps. First, the Euclidean distance matrix for each cluster and its mean distances are computed. The density is then computed as a fraction of the number of minority samples within the cluster and the average Euclidean distance power to the number of features m:

$$density(c) = \frac{minority\_count(c)}{avg\_minority\_distance(c)^m}.$$

Sparsity is the inverse of density:

$$sparsity(c) = \frac{1}{density(c)}.$$

The sampling weight is calculated as a fraction of the sparsity and the sum of all sparsities:

$$sampling\_weight(c) = \frac{sparsity(c)}{\sum_{f \in filtered\_clusters} sparsity(f)}.$$

[4]`https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.KMeansSMOTE.html`

The number of synthetic samples to be generated in each cluster is counted as the multiplication of the sampling weight and the total number of samples to be generated G:

$$number\_of\_samples(c) = G \cdot sampling\_weight(c).$$

The G samples are generated using the modified SMOTE algorithm, and synthetic samples are added to the original training set.

3.6 CTGAN

CTGAN [35] represents the GAN-based techniques. It is a generative method, not concentrating only on the minority class oversampling but rather on the problem of having imbalanced categorical features. Its implementation comes from a library of the same name[5] and consists of three main classes: `DataTransformer`, `DataSampler`, and `CTGAN`.

The `DataTransformer` is responsible for data transformation into the representation that is suitable for GAN. For continuous features, a mode-specific normalization is used, and for categorical features, one-hot encoding is used. The transformer is fitted individually for each feature, and training data are transformed feature by feature. The advantage is that the transformation is reversible. After sampling, the synthetic data can be converted back into the original feature space.

The `DataSampler` uses the transformed data for sampling a conditional vector and real data sampling based on the conditional vector. The sampling of a training conditional vector is done by randomly selecting a categorical feature and selecting a category within the feature based on the logarithm of its frequency. This so-called training-by-sampling ensures the examination of all values within a categorical feature [51].

During the training, CTGAN performs several actions in each step of the epoch. For the discriminator update, fake samples are generated from Gaussian noise and are concatenated with the sampled conditional vector. Real data are sampled based on the conditional vector. A discriminator is trained on the real and fake samples. CTGAN uses Wasserstein loss with gradient penalty (Eq. 2.4). In the same way, fake samples are created also for a generator update. The generator loss is counted as a Wasserstein loss plus conditional loss. Conditional loss penalizes the generator for creating arbitrary data given a conditional vector. It is computed as a cross-entropy on the conditioned categorical feature.

The architecture of both the generator and discriminator consists of two fully connected hidden layers and an output layer. Discriminator architecture was inspired by PacGAN [52], which means that it makes decisions based on packs of given samples. The activation function of the generator's output layer differs based on the feature type. It can be either *tanh* for the normalized values or *Gumbel-softmax* (Eq. 2.3) for the one-hot vectors. In the hidden layers of the generator, $ReLU$ is used as an activation function and in the discriminator's hidden layers, $LeakyReLU$ is used.

CTGAN has many parameters, such as the parameters of Adam optimizers for the generator and the discriminator, the number of epochs, batch size, and the dimensions of the network layers.

3.7 CTAB-GAN

CTAB-GAN implementation is also available on GitHub[6]. CTAB-GAN was designed to handle continuous, categorical, and mixed data types. The mixed data type combines a continuous feature with special discrete values. Having a dataset, the information about each feature type

[5]`https://github.com/sdv-dev/CTGAN`
[6]`https://github.com/Team-TUD/CTAB-GAN/tree/main`

has to be given externally. The difference from CTGAN is in the conditional vector. Not only categorical features but all features are used as a part of it.

The implementation of CTAB-GAN was inspired by previous works. It also contains `DataTransformer` but adds a Mixed-type Encoder for mixed features. However, CTAB-GAN has a more complicated update. Next to the adversarial loss, there are three more loss functions: classification, information, and generator loss. An auxiliary classifier is added parallel to the discriminator to calculate the classification loss that is computed as the binary cross entropy between the real and predicted class labels to force the generator to create semantically correct samples. Information loss is a loss between real and generated data. Concretely, it compares their means and standard deviations. Generator loss is computed as a cross-entropy between the generated output values and the sampled conditional vector to ensure that the generator creates samples as conditioned by the conditional vector. The generator objective function is calculated as follows:

$$\mathcal{L}^G = \mathcal{L}^G_{orig} + \mathcal{L}^G_{info} + \mathcal{L}^G_{class} + \mathcal{L}^G_{generator}$$

whereas the discriminator objective function stays unchanged: $\mathcal{L}^D = \mathcal{L}^D_{orig}$.

The architecture of CTAB-GAN adopts CNN in the generator and discriminator. Tabular data are, therefore, transformed into an image domain to be fed into CNN. Classifier network, on the other hand, contains fully connected layers with *LeakyReLU* as an activation function in all four hidden layers and *sigmoid* in the output layer.

CTAB-GAN takes as a parameter the path to the `csv` file containing the training dataset; therefore, for each experiment in this thesis, the training data were stored before oversampling. As mentioned earlier, the types of features have to be specified externally. CTAB-GAN takes `categorical_columns`, `integer_columns`, `mixed_columns` and `log_columns` as parameters. The last parameter is used for features with skewed exponential distribution. Another parameter is `problem_type` as CTAB-GAN can be used not only for the binary classification problem but also for regression tasks.

3.8 TVAE

TVAE [35] was introduced in the same article as CTGAN, and its implementation also comes from the same library `ctgan`. Both encoder and decoder architectures consist of fully connected layers.

The encoder consists of 2 hidden layers with *ReLU* activation function and a dimension 128. There are two output layers, one representing the mean and one for the standard deviation, without an activation function.

The decoder also consists of 2 hidden layers with *ReLU* activation function and a dimension of 128. The activation function differs for the output layer based on the feature type. It can be either *softmax* for the one-hot vectors or *tanh* for the standardized value of the continuous features within the mode.

The same `DataTransformer` is used as in the case of CTGAN. The dimensions of each hidden layer can be adjusted through parameters `compress_dims` and `decompress_dims` of TVAE, the same as the latent space dimension `embedding_dim`. Other parameters of the model are the weight decay for the Adam optimizer, factor used in the objective function, the number of epochs, and batch size.

TVAE is used for oversampling, for example, by Kim et al. [32], who train the model only on the minority class. In contrast, in the work by Darabi et al. [45], CTGAN, which comes from the same library, is trained on the whole dataset, and sampling is done by generating synthetic samples until the needed amount of minority class samples is obtained. In our work, we used the latter approach for CTGAN, CTAB-GAN, and TVAE. However, we encountered a problem with *haberman* dataset, as it is the smallest used. TVAE was not able, in some cases, to learn to generate minority-class data.

3.9 TAEI

TAEI [45] implementation comes from the library `sagemaker-scikit-learn-extension`[7]. This method can be implemented as an instance of `LatentSpaceOversampler`, which takes `model` and `base_oversampler` as parameters. We have chosen VAE as the model because AE and VAE perform the best, as stated in the original article [45], and SMOTE as the oversampler. However, as the article says, any interpolation method can be used. For VAE, the features must be divided into categorical and continuous and fed into the model separately in parameters `categorical_features` and `continuous_features`. For the categorical features, TAEI also takes the dimensions of the features as the parameter `categorical_dims`. The `sampling_strategy` is a parameter of the oversampler that specifies which classes should be sampled. In our experiment, it is set to `minority` to get a balanced dataset.

The steps for TAEI are using an encoder to map data to a dense latent space, interpolating data in the latent space using SMOTE or other interpolation methods, and using a decoder to map data back to the original feature space. What is more, categorical features have to be transformed before being used as input to the encoder. Therefore, the embedding layer is applied first on each sample that is represented by a vector created by concatenation of all features. This embedding layer's output is a vector that serves as an input to the encoder.

During the training, reconstruction loss is minimalized. Having both continuous and categorical features, the reconstruction loss consists of the sum of mean squared error (MSE) and softmax loss:

$$J_{recon}(D;\theta,\phi) = \sum_{x_i}\sum_{c}^{|C|} ||h_\theta(z_i)^c - x_i^c||_2^2 + \alpha \sum_{x_i}\sum_{t}^{|T|}\sum_{o} \mathbf{1}[x_i^t = o]\log(h_\theta(z_i)^o)$$

where x is the input vector, z is the encoder output $z = g_\phi(x)$, $h_\theta(z)$ represents the decoder mapping, and C and T represent the continuous and categorical features.

After AE training, the oversampler is used for generating synthetic data in the latent space. Those generated samples are then mapped with the decoder back to the feature space.

3.10 SMOTified-GAN

SMOTified-GAN [49] is not implemented as a part of any library. However, the implementation from its authors can be found on GitHub [8].

The algorithm has two steps. First, SMOTE is used to oversample the minority class data the same way as described in Section 3.1. The second step is training GAN with the training minority class data and oversampled minority data from SMOTE.

The generator consists of 4 blocks of fully connected layers with batch normalization, *ReLU* activation function and increasing output dimensions. The last layer is also a fully connected layer but without batch normalization and with *sigmoid* as the activation function. Discriminator's architecture consists of 3 fully connected hidden layers with *LeakyReLU* activation function and shrinking dimensions. The output layer is also a fully connected layer with *sigmoid* as the activation function.

[7] `https://github.com/aws/sagemaker-scikit-learn-extension/tree/master/src/sagemaker_sklearn_extension/contrib/taei`

[8] `https://github.com/sydney-machine-learning/GANclassimbalanced/tree/main`

The objective function of this model is binary cross-entropy between the true and predicted values. As the optimizer, Adam is used with a learning rate of 0.00001, as stated in the original article [49].

3.11 Summary

In the experimental part of this thesis, we have chosen 10 oversampling techniques with publicly available implementations, concretely 5 traditional ones, 2 GAN-based, 1 AE-based, and 2 combined. Traditional and combined techniques usually need preprocessing, where features are transformed using one-hot encoding for categorical features and standardization for continuous features as described in more detail in the following Chapter 4, whereas generative-based techniques handle preprocessing within their implementations. For a fair comparison, the hyperparameters of the methods are set to default values or to values stated in their corresponding articles, and they are not tuned.

Chapter 4

Implementation

This Chapter describes the implementation details and the experiments. Firstly, the selected datasets with preprocessing, classifiers, and metrics that are used in the experiments are outlined, followed by the description of the experiment setup.

4.1 Datasets

For the experiments, 9 datasets were selected. Those datasets appear repeatedly in the reviewed articles and were chosen such that they have different IR, number of features, and different data types. They come from KEEL[1] [53], `imblearn` library[2], and UCI Machine Learning Repository[3] [54]. Concretely 6 datasets are coming from KEEL: *pima*, *haberman*, *abalone9-18*, *yeast3*, *abalone20-vs-8-9-10*, and *winequality-red-4*, 2 from UCI: *adult* and *german*, and 1 dataset comes from `imblearn`: *mammography*.

4.1.1 Preprocessing

Preprocessing is an integral part of ML projects working with datasets. The preprocessing steps are, for example, handling noisy data, imputing null values, removing columns that are not essential for the ML task, standardization or normalization, and transformation into the embedding suitable as an input for a machine learning model.

KEEL datasets follow a structure where the target feature is always called `Class`. Therefore, all datasets are adjusted so that they match this pattern. What is more, the values of the target feature are `negative` and `positive` for all KEEL datasets. However, as `Class` represents a binary feature, those values are transformed into numbers 0 and 1.

For simplicity, we have decided to drop rows containing null values instead of their imputation as their amount is not significant, and they are mostly coming from the majority class. Duplicates of rows and constant columns are also removed as they do not bring any new information to the data. The specifications of each preprocessed dataset can be seen in Table 4.1.

The datasets consist of numerical as well as categorical features. Both types require preprocessing before the dataset can be used by the oversamplers and classifiers. Numerical features are scaled using `StandardScaler`[4] that for each feature independently subtracts mean μ from

[1]`https://sci2s.ugr.es/keel/datasets.php`
[2]`https://imbalanced-learn.org/stable/references/datasets.html`
[3]`https://archive.ics.uci.edu/datasets`
[4]`https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler`

Name	Shape	Data types	Imbalance Ratio
pima	(768, 9)	num: 9	1.866
german	(1000, 21)	cat: 13, num: 8	2.333
haberman	(289, 4)	num: 4	2.658
adult	(45175, 15)	cat: 8, num: 7	3.033
yeast3	(1453, 9)	num: 9	7.969
abalone9-18	(731, 9)	num: 8, cat: 1	16.405
winequality-red-4	(1359, 12)	num: 12	24.642
mammography	(7849, 7)	num: 7	29.902
abalone20-vs-8-9-10	(1916, 9)	cat: 1, num: 8	72.692

Table 4.1 Datasets Specification: name of the datasets, shape (number of samples, number of features), data types (numerical and categorical), and imbalance ratio

the initial value x and divides this difference by standard deviation σ resulting in a value of a zero mean and unit variance:

$$z = \frac{x - \mu}{\sigma}.$$

The categorical features are encoded using `OneHotEncoder`[5] to a one-hot vector. `OneHotEncoder` has a parameter `handle_unknown` which specifies what should be done when an unknown category appears in data transformed by this encoder. This can happen during encoding test data with an encoder fitted to training data. We have chosen to set the value of this parameter to *ignore*, which means that the one-hot vector for the unknown categories will contain only zeros and in the inverse transformation, the value of the category will be `None`. Both `StandardScaler` and `OneHotEncoder` come from `sklearn` library.

In the researched works, preprocessing steps differ and are not the same for all, using, for example, standardization or normalization into $[-1, 1]$ or $[0, 1]$ for continuous features and one-hot, label, or ordinal encoding for categorical features, making their results incomparable. What is more, some works do not describe the preprocessing steps at all.

However, standardization and encoding are not needed for all oversampling techniques used in this thesis. Some of them handle preprocessing by themselves. Concretely, standardization and one-hot encoding are done for SMOTE, Borderline-SMOTE, Polynomial-Fit-SMOTE, k-means-SMOTE, LoRAS, and SMOTified-GAN. CTGAN, TVAE, and CTAB-GAN implementations contain `DataTransformer`, which is responsible for correct data transformation, as stated in Chapter 3. TAEI accepts both numerical and categorical data types. However, the categorical features must be transformed with `OrdinalEncoder` before they are used by the model and the continuous features are preprocessed as in the case of traditional oversampling techniques.

4.2 Evaluation Metrics

Several metrics exist for model evaluation, but not all of them are suitable for learning from imbalanced datasets. Some of the commonly used metrics are overall accuracy, balanced accuracy, f1-score, G-mean, ROC, and AUC [3].

Most metrics values are calculated from the elements of the confusion matrix that can be seen in Figure 4.1.

Overall accuracy

Overall accuracy[6] is the ratio of the number of correctly classified samples and the number of

[5]`https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder`
[6]Whenever only accuracy is used in this work, it denotes overall accuracy.

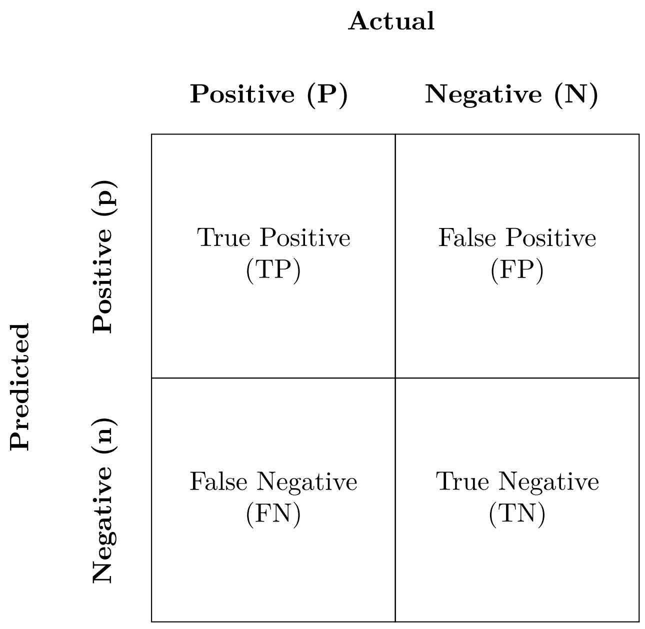


Figure 4.1 Confusion Matrix

all samples [19].

$$OA = \frac{TP + TN}{TP + TN + FP + FN}$$

This metric is very popular. However, it is not suitable for an imbalanced dataset. For example, having a test dataset with 95 majority and 5 minority samples, the classification model can classify all samples as majority ones, and the OA would still be high, concretely 95 %.

Balanced accuracy
Balanced accuracy (also called average accuracy) is a more suitable metric for imbalanced domains.

$$BA = \frac{1}{2} \cdot (\frac{TP}{TP + FN} + \frac{TN}{TN + FP})$$

If a classifier favours one class, BA drops because the performance on the other class is not accurate. On the contrary, OA stays high even if one class is favoured [3].

F1-score
F1-score represents a harmonic mean of precision and recall.

$$f1\text{-}score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

where precision and recall are calculated as follows:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

F1-score balances those two metrics and tries to keep both of them high [8].

G-mean
G-mean represents a geometric mean. This metric is more suitable for imbalanced domains than OA, as it is counted as the root of the product of sensitivity and specificity. G-mean tries to keep accuracies of both classes balanced [3].

$$G\text{-}mean = \sqrt{\frac{TP}{TP+FN} \cdot \frac{TN}{TN+FP}}$$

ROC curve
ROC curve measures the performance of a classifier considering all possible trade-offs between true positive rate (TPR) and false positive rate (FPR) [3].

$$TPR = \frac{TP}{TP+FN}$$

$$FPR = \frac{FP}{FP+TN}$$

AUC
The Area Under The ROC Curve (AUC) is a scalar value representing and summarizing the ROC curve. Its advantage is that AUC is not biased towards any class.

Having an imbalanced problem, TPR is usually the most important metric, as the minority class is often the important one. A suitable metric should be chosen based on the problem. When both classes are supposed to be identified correctly, BA or G-Mean should be selected. If the goal is to have the highest accuracy, OA can be used. However, there is a chance of poor performance on the minority class.

In this thesis, we have decided to measure both balanced and overall accuracies. OA may drop after oversampling because, having an imbalanced dataset, classifiers can be biased towards the majority class and perform poorly on the minority class, which cannot be captured by this metric. We also measure the f1-score and its components, precision and recall, since the f1-score is a suitable metric for learning from imbalanced datasets, and values of its components can give us additional information about the performance. The last measured metric we use is AUC.

4.3 Classifiers

The selection of a classification model plays a big role in the imbalance dataset problem. As many evaluation metrics exist, there are also several classifiers that can be used for a binary classification. However, some of them are more sensitive to class imbalance than others.

In the work by Gala [55], Logistic Regression (LOGIT), Decision Tree Classifier (DTC), and Random Forest Classifier (RFC) are used, and the results show that those learners are sensitive to class imbalance. For this reason, those models are also used in this work.

The results in the work by Lemnaru et al. [3] indicate that the Support Vector Machine (SVM) is less sensitive to the IDS problem and, as the previous study has shown, DTC is sensitive to the class imbalance. MLP is less sensitive than DTC but more sensitive than SVM.

The reason why DTC is so popular considering the IDS problem is its simplicity and the fact that it is built greedily and is not resistant to overfitting. That means that if oversampling together with DTC results in a good performance, the quality of the generated data is high.

Another classification model is k Neighbors Classifier (KNN), and it is also sensitive to the quality of the generated samples. As a distance-based learner, the distance between synthetic samples and samples in the test set is essential. The distance is the measure of how well the generated samples approximate the minority class distribution.

Based on the research done as a part of this thesis, 6 repeatedly used classification models, concretely Decision Tree Classifier, Random Forest Classifier, Logistic Regression, Support Vector Machine, Multilayer Perceptron, and k Neighbors Classifier are used for evaluation in the practical part.

4.4 Experiment Setup

As part of this thesis, 10 existing oversampling techniques introduced in Chapter 3 and 1 novel method presented in the following Chapter 5 are implemented and used for oversampling the training dataset for the downstream binary classification task of 9 tabular datasets. Together, we have 11 oversampling techniques, 9 datasets, 6 classifiers, and 6 evaluation metrics. For the implementation, the Python programming language is used.

First of all, we have to get and preprocess datasets described in Section 4.1. After the data preparation, for each oversampling technique, one core function `scoring_dfs` is called with a difference in the parameter `oversampling_func`, which returns the oversampled training dataset. Before the data are oversampled, each dataset is split into train and test sets in a stratified way to ensure the same data distributions of both sets. The split ratio is $70 - 30$.

Subsequently, only the training dataset is oversampled. For each oversampling technique, we measure the time of data generation in seconds. In the case of methods that need training, we measure this time as well because it is part of the oversampling process. Methods like CTGAN, TVAE, and CTAB-GAN generate both minority and majority samples. Thus, the class label is added as a categorical feature, and after the sampling, only synthetic minority samples are retained. This is repeated until there are sufficient minority-class samples. The same approach was used in the work by Darabi et al. [45].

As mentioned in Chapter 3, some methods like CTGAN and CTAB-GAN handle the data transformation themselves, and as the oversampling result, we get data in their original input format. Therefore, after the data balancing, we use the same standardization and one-hot encoding as before the oversampling in the case of SMOTE, Borderline-SMOTE, etc., to get a fair comparison of the techniques in the downstream tasks.

After having a balanced training dataset, we can measure the performance of the classification models. The scoring function takes as a parameter flag, which denotes whether classifiers' hyperparameters (HPs) should be tuned or left with default values. The HPs are tuned using grid search that does a complete search over a given grid of parameters. Grid search is done using 10-fold stratified cross-validation over the training dataset and f1-score as the evaluation metric. The values used for each classification model in the grid search can be found in Appendix C.

HPs of the oversampling techniques are not tuned and are used with their default values or with values stated in the corresponding articles. For SMOTE, it means, for example, setting `k_neighbors` to 5. The same value is the default in all other traditional oversampling techniques used in our experiment.

After having a classifier with default or tuned HPs, it is fitted to the whole training set, and predictions are made using the test set. Eventually, each of the 6 metrics is evaluated on the predictions and true labels.

All those values, namely, oversampled data, oversampling duration, classifiers' parameters, obtained results, and classification reports, are stored during the algorithm run for evaluation and comparison. A pseudo-code of the simplified scoring function without values storing can be seen in Algorithm 1.

Algorithm 1 Scoring function

```
procedure SCORING_DFS($dfs, clfs, metrics, default\_clf, oversampling\_func, transform$)
    for $df \in dfs$ do
        $X\_train, y\_train, X\_test, y\_test \leftarrow df$
        $X\_train, y\_train \leftarrow oversampling\_func(X\_train, y\_train)$
        if $transform$ then
            $X\_train, X\_test$ $\leftarrow$use one-hot encoding and standardization of $X\_train, X\_test$
        end if
        for $clf\_name \in clfs$ do
            if $defaulf\_clf$ then
                $classifier \leftarrow$ create classifier of $clf\_name$ with default HPs
            else
                $params \leftarrow$ get HPs grid for classifier with $clf\_name$
                $classifier \leftarrow grid\_search(X\_train, y\_train, params, clf\_name, Metrics.f1)$
            end if
            $classifier.fit(X\_train, y\_train)$
            $predicted \leftarrow classifier.predict(X\_test)$
            for $metric \in metrics$ do
                $score \leftarrow metric(y\_test, predicted)$
            end for
        end for
    end for
    return
end procedure
```

Chapter 5

Novel Method

In this Chapter, our proposed oversampling technique for the imbalanced tabular dataset of binary classification problem is presented. The inspiration for its structure is given here, same as details about the data preprocessing, architecture, training and synthetic data generation.

5.1 Inspiration

In Chapter 2, techniques combining traditional and generative-based methods are introduced, concretely TAEI and SMOTified-GAN. The idea of TAEI is to use an encoder to map data into the latent space, oversample in the dense latent space using an arbitrary interpolation technique, such as SMOTE or Poly, and map the interpolated samples back to the feature space. The idea of SMOTified-GAN is that SMOTE can create samples that do not match the real data distribution. Therefore, GAN takes the oversampled minority class samples as input and transforms them to match the minority class distribution.

Our method combines those two ideas. VAE, together with an interpolation method, is used to create synthetic minority samples as in TAEI. Those samples are used as input to GAN to transform them in a similar way as in SMOTified-GAN.

The idea is that oversampling in a dense latent space creates better synthetic samples than in sparse high-dimensional space, and the quality of those samples is further improved by transformation using GAN to match the minority class distribution. We call our method Latent Interpolation Tabular GAN (LIT-GAN).

5.2 Preprocessing

Tabular datasets often include a mix of continuous and categorical features. Inspired by CTGAN and CTAB-GAN, LIT-GAN uses `DataTransformer`, which is responsible for data transformation for both types.

The `DataTransformer` is fitted to the training data, and each feature is handled independently by the transformer. Continuous features are transformed using VGM, and the result for each value consists of two parts: a one-hot vector representing a mode and a normalized value within the mode. The default maximal number of modes is set to 10, the same as in CTGAN and CTAB-GAN. Categorical features are transformed using one-hot encoding. `ClusterBasedNormalizer` and `OneHotEncoder` from Reversible Data Transforms (`rdt`[1]) library are used. Both transformers

[1] `https://docs.sdv.dev/rdt/#owned-and-maintained-by-datacebo`

also implement a method for a reverse transformation. The reverse transformation is used after synthetic data are sampled by the generator in the oversampling process.

Compared to TAEI, the data preprocessing differs. TAEI, in its original article [45], uses OrdinalEncoder for categorical features and normalization into $[-1, 1]$ for continuous features. In the original article of SMOTified-GAN [49], the preprocessing steps are not described, only mentioned as "basic preprocessing steps".

5.3 Architecture

LIT-GAN consists of 3 main parts: VAE, oversampler, and GAN. VAE and GAN can be further divided into encoder *Enc* and decoder *Dec*, respectively, generator G and discriminator D. VAE is optimized to generate data similar to the training data, and GAN is optimized to transform the output of VAE to match minority class distribution. Our implementation of LIT-GAN is done using `tensorflow`[2] library.

The oversampler is given as a parameter of LIT-GAN, and it can be an arbitrary interpolation technique that implements function `sample(X,y)`, as the oversampling methods in the `smote-variants` library, which we use in our model. In the experiment, we use SMOTE for the comparison with TAEI and SMOTified-GAN.

VAE architecture was inspired by TVAE [35], as we use the same preprocessing techniques. Both *Enc* and *Dec* networks contain two fully connected hidden layers with *ReLU* activation function. The encoder's output layers are also fully connected layers, mapping the output of the last hidden layer to the latent space.

The data are transformed as described in the previous section before they are used as inputs for the encoder. N_c is the number of continuous features, and N_t is the number of categorical features. The transformed data structure looks as follows: $x = \alpha_1 \oplus \beta_1 \oplus \ldots \oplus \alpha_{N_c} \oplus \beta_{N_c} \oplus \mathbf{t}_1 \oplus \ldots \oplus \mathbf{t}_{N_t}$. α_i represents the normalized continuous value within the mode after applying the normalization on the i-th continuous feature. β_i represent the one-hot vector for m_i modes of the i-th continuous feature. $\mathbf{t}_i$ is the one-hot vector for the i-th categorical feature, and $\oplus$ denotes concatenation of the values.

Having training data x, the architecture of the encoder is:

$$\begin{cases} h_0 = x \\ h_1 = \texttt{ReLU}(\texttt{FC}_{|x|\to 128}(h_0)) \\ h_2 = \texttt{ReLU}(\texttt{FC}_{128\to 128}(h_1)) \\ \mu = \texttt{FC}_{128\to 32}(h_2) \\ \sigma = exp(\frac{1}{2} \cdot \texttt{FC}_{128\to 32}(h_2)) \end{cases}$$

Having z value counted as:

$$z = \mu + \epsilon \cdot \sigma$$

where $\epsilon \sim N(0, I)$, the decoder reconstructs data from the latent space into the feature space. Its architecture of the hidden layers is the opposite of the encoder's. The whole decoder network looks as follows:

$$\begin{cases} h_0 = z \\ h_1 = \texttt{ReLU}(\texttt{FC}_{32\to 128}(h_0)) \\ h_2 = \texttt{ReLU}(\texttt{FC}_{128\to 128}(h_1)) \\ \alpha'_i = \texttt{tanh}(\texttt{FC}_{128\to 1}(h_2)) & 1 \le i \le N_c \\ \beta'_i = \texttt{softmax}(\texttt{FC}_{128\to m_i}(h_2)) & 1 \le i \le N_c \\ \mathbf{t}'_i = \texttt{softmax}(\texttt{FC}_{128\to |T_i|}(h_2)) & 1 \le i \le N_t \end{cases}$$

[2]`https://www.tensorflow.org/`

The architecture differs from TVAE in the dimensions of the layers.

GAN architecture was inspired by CTGAN and consists of two networks: generator G and discriminator D. Both are fully connected multi-layer networks. In the case of the generator, there are 2 hidden layers with batch normalization and $ReLU$ activation function and an output layer that differs in the activation function based on the feature type. The discriminator contains 2 hidden layers with $LeakyReLU$ activation function and dropout and output layer without an activation function. The dimensions of the hidden layers of the generator and discriminator are the same, while the output layer of G has the same dimension as its input, and the D output layer has a dimension of 1.

Formally, having VAE's decoder output representing a synthetic minority class sample $x'_V = \alpha'_1 \oplus \beta'_1 \oplus \ldots \oplus \alpha'_{N_c} \oplus \beta'_{N_c} \oplus \mathbf{t'}_1 \oplus \ldots \oplus \mathbf{t'}_{N_t}$, the generator architecture is:

$$\begin{cases} h_0 = x'_V \\ h_1 = \texttt{ReLU}(\texttt{BN}(\texttt{FC}_{|\hat{x}_V| \to 256}(h_0))) \\ h_2 = \texttt{ReLU}(\texttt{BN}(\texttt{FC}_{256 \to 256}(h_1))) \\ \hat{\alpha}_i = \texttt{tanh}(\texttt{FC}_{256 \to 1}(h_2)) & 1 \leq i \leq N_c \\ \hat{\beta}_i = \texttt{gumbel}_{0.2}(\texttt{FC}_{256 \to m_i}(h_2)) & 1 \leq i \leq N_c \\ \hat{\mathbf{t}}_i = \texttt{gumbel}_{0.2}(\texttt{FC}_{256 \to |T_i|}(h_2)) & 1 \leq i \leq N_t \end{cases}$$

The discriminator architectures for generator's output $\hat{x}_G = \hat{\alpha}_1 \oplus \hat{\beta}_1 \oplus \ldots \oplus \hat{\alpha}_{N_c} \oplus \hat{\beta}_{N_c} \oplus \hat{\mathbf{t}}_1 \oplus \ldots \oplus \hat{\mathbf{t}}_{N_t}$ is:

$$\begin{cases} h_0 = \hat{x}_G \\ h_1 = \texttt{drop}_{0.5}(\texttt{LeakyReLU}_{0.2}(\texttt{FC}_{|\hat{\mathbf{x}}_\mathbf{G}| \to 256}(h_0))) \\ h_2 = \texttt{drop}_{0.5}(\texttt{LeakyReLU}_{0.2}(\texttt{FC}_{256 \to 256}(h_1))) \\ \mathcal{D}(\cdot) = \texttt{FC}_{256 \to 1}(h_2) \end{cases}$$

In our network, the generator does not sample from Gaussian noise but takes synthetic samples generated by VAE as an input. What is more, the real data are not sampled from the whole training dataset but only from the minority class.

5.4 Tuning

Our method has many parameters that have to be set properly in order to perform well. Those parameters are optimizer, learning rates, dimensions of network layers, and number of epochs for VAE and GAN training. The values of those parameters were set based on the results of grid search using 5-fold cross-validation on training data from *abalone9-18* dataset with LOGIT as the classification model and f1-score as the evaluation metric.

Our method consists of 2 parts: VAE and GAN. As there are many combinations of those parameters, we first searched the space of parameters for VAE and then for GAN with the parameters of VAE set by the first step. Finally, having an idea of a possible smaller amount of suitable parameters for both parts, we searched for the parameters of the whole network. As an optimizer, Adam is used with a learning rate of 0.0001. The number of epochs for VAE is set to 300, whereas for GAN, it is 500. The selected dimensions of the layers can be seen in the previous Section 5.3.

5.5 Training

Training consists of 2 main parts. First, VAE is learned to reconstruct the whole training dataset. In each epoch, batches of the training dataset are passed through VAE one by one, and the entire

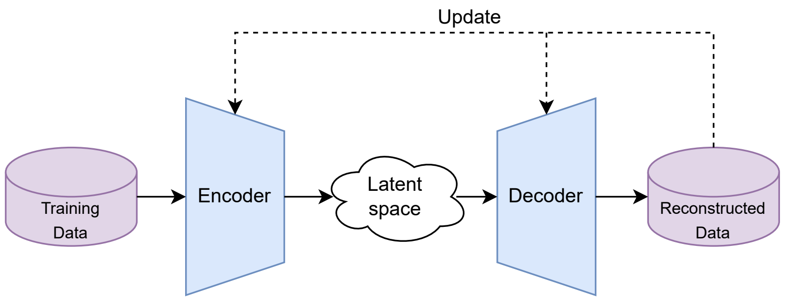


Figure 5.1 LIT-GAN: VAE Training

VAE network is trained by optimizing the reconstruction loss and Kullback-Leibler divergence:

$$KLD = \frac{1}{2} \sum_{i}^{|batch|} (\mu_i^2 + \sigma_i^2 - 1 - \log(\sigma_i^2))$$

where μ and σ are values obtained from the encoder.

The VAE training procedure is outlined in Algorithm 2 and in Figure 5.1.

Algorithm 2 LIT-GAN: VAE Training

```
procedure TRAIN_VAE($train\_data$)
    $batches \leftarrow$ create batches from $train\_data$
    for $epoch \in vae\_epochs$ do
        for $batch \in batches$ do
            $mu, std, logvar \leftarrow encoder(batch)$
            $eps \leftarrow$ sample from $N(0, I)$
            $z \leftarrow eps \cdot std + mu$
            $reconstruction, sigmas \leftarrow decoder(z)$
            $loss \leftarrow$ calculate loss
            compute and apply gradients
        end for
    end for
    return
end procedure
```

The second phase is GAN training. The generator and discriminator are trained as a traditional unconditional GAN with Wasserstein loss with a gradient penalty (Eq. 2.4). In each epoch, the encoder is applied to the whole training dataset to map it into the latent space. There, an oversampler, such as SMOTE, is used to generate minority samples, which are mapped back to the original space using the decoder. Those synthetic samples are divided into batches, and each batch is used as input for the generator. The same number of observations is randomly sampled from the training data of the minority class. The output of the generator and the sampled minority data are then used as input for the discriminator. The goal is for the generator to produce samples that are indistinguishable from the real minority samples. GAN training procedure can be seen in Algorithm 3 and in Figure 5.2.

Compared to SMOTified-GAN, LIT-GAN in each GAN training epoch uses its VAE and oversampler to create minority-class synthetic samples, whereas SMOTified-GAN uses the same synthetic samples across all epochs. We have chosen our approach based on the similarity with data generation from noise, where the noise is sampled also in each epoch.

Finally, the whole LIT-GAN training procedure can be seen in Algorithm 4.

Algorithm 3 LIT-GAN: GAN Training

```
procedure TRAIN_GAN($train\_data, y$)
    for $epoch \in gan\_epochs$ do
        $mu \leftarrow encoder(batch)$
        $x\_sampled, y\_sampled \leftarrow oversampler(mu, y)$
        $x\_decoded \leftarrow decoder(x\_sampled)$
        $batches \leftarrow$ create batches from $x\_decoded$
        for $batch \in batches$ do
            $fakes \leftarrow generator(batch)$
            $reals \leftarrow$ sample minority data from $train\_data$
            $y\_fake \leftarrow discriminator(fakes)$
            $y\_real \leftarrow discriminator(reals)$
            $loss \leftarrow$ calculate loss for generator and discriminator
            compute and apply gradients
        end for
    end for
    return
end procedure
```

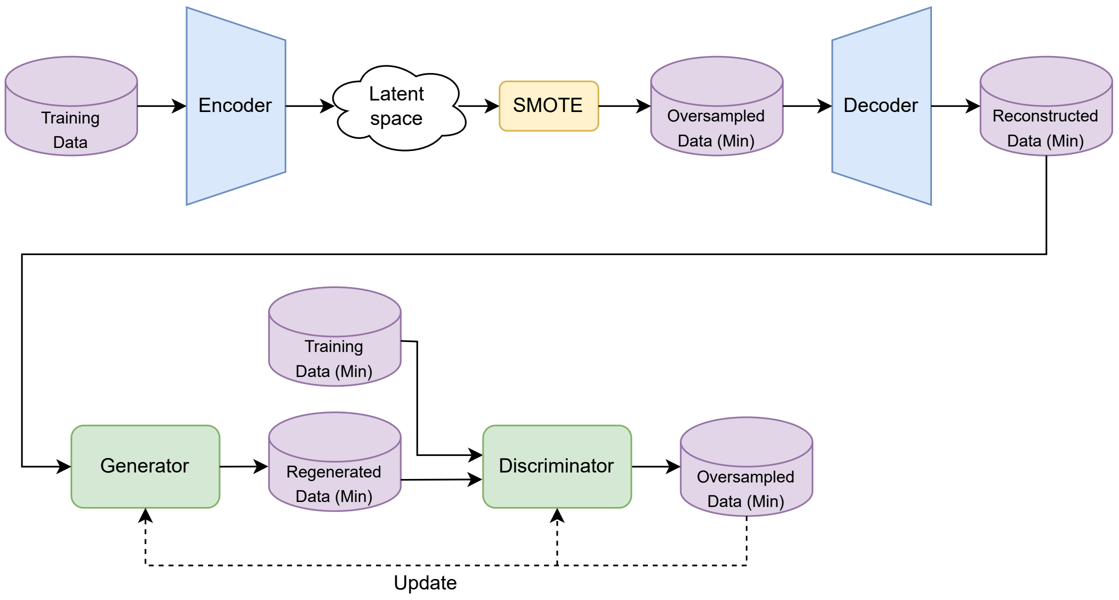


Figure 5.2 LIT-GAN: GAN Training: (Min) denotes data belonging to the minority class.

Algorithm 4 LIT-GAN Training

```
procedure TRAIN($X, y, categorical\_columns$)
    $train\_data \leftarrow$ transform $X$ with DataTransformer
    $train\_data\_dim \leftarrow$ dimension of the $train\_data$
    initialize encoder, decoder, generator, discriminator with correct dimensions
    $train\_vae(train\_data)$
    $train\_gan(train\_data, y)$
    return
end procedure
```

5.6 Sampling

Sampling of the synthetic data is done after the whole network consisting of VAE and GAN is trained. The encoder is used to map the train data into the latent space. In the latent space, the oversampler creates synthetic minority samples that are mapped back to the original space with the decoder. Next, the generator is applied to the output of the decoder to create final synthetic minority samples. The last step is transforming those samples back to the feature space with a reverse transformation of the `DataTransformer` instance fitted to the training dataset, as described in Section 5.2. The sampling procedure can be seen in Algorithm 5.

Algorithm 5 LIT-GAN Sampling

```
procedure SAMPLE
    $train\_data \leftarrow$ transformed training data with DataTransformer
    $mu \leftarrow encoder(train\_data)$
    $x\_sampled, y\_sampled \leftarrow oversampler(mu, y)$
    $x\_decoded \leftarrow decoder(x\_sampled)$
    $x\_synthetic \leftarrow generator(x\_decoded)$
    return $x\_synthetic, y\_sampled$
end procedure
```

Chapter 6

Results

In this Chapter, the results of the experiments are presented. The methods are compared based on their duration and performance using different classification methods and evaluation metrics on the selected datasets that are described in Chapter 4. In some of the presented tables, the names of the oversampling techniques and datasets had to be shortened due to space limitations.

6.1 Performance

In this Section, we present the results of evaluating the classification models on all 9 datasets. The measurement was first done on the original imbalanced datasets and then on the datasets oversampled by techniques presented in Chapter 3.

As mentioned in Chapter 4, the scoring function has a parameter which decides if each classifier should be used with default hyperparameters or tuned through grid search. We have run the oversampling 3 times with different random seeds for the train-test split. Then, the classifiers were fitted to those oversampled datasets with HPs tuning and without HPs tuning for all 3 runs, and the results were averaged.

6.1.1 Results Without Tuned Classifiers Hyperparameters

By running the program more times with a different random seed, we retrieved the mean and standard deviation of the results. This is done as follows: we take the results from all runs for each dataset, the oversampling technique, and the classification method separately, and then we compute the mean and standard deviation. For each technique and dataset, the result of the classifier with the best average score is taken.

The average f1-score and accuracy of those runs for each technique and dataset can be seen in Tables 6.1 and 6.2. Results for all other measured metrics are in Appendix B. Only the result of the best-performing classifier, together with the mean value, standard deviation, and the difference from the best average result of a classification done using the original imbalance dataset, are shown. The datasets are sorted by their imbalance ratios in ascending order. The imbalance ratios can be found in Table 4.1. The highest score for each dataset is highlighted.

It can be seen that we do not have the result for the combination of k-means-SMOTE and *abalone9-18* dataset. This is caused by the problem described in Section 3.5. The oversampling resulted in an error while creating the clusters. Even a high number of clusters did not help to overcome this problem.

dataset	none	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
pima	MLP 0.658 ±.028	MLP 0.657 ±.042 (-0.001)	SVM 0.687 ±.033 (0.029)	LOGIT 0.686 ±.05 (0.028)	**MLP** **0.691** **±.037** **(0.033)**	MLP 0.664 ±.033 (0.006)	SVM 0.679 ±.044 (0.021)	LOGIT 0.665 ±.043 (0.007)	LOGIT 0.678 ±.033 (0.02)	SVM 0.675 ±.058 (0.017)	RFC 0.689 ±.042 (0.031)	MLP 0.681 ±.043 (0.023)
german	MLP 0.522 ±.027	LOGIT 0.571 ±.027 (0.049)	SVM 0.591 ±.026 (0.069)	**SVM** **0.61** **±.019** **(0.088)**	LOGIT 0.582 ±.034 (0.06)	LOGIT 0.5 ±.0 (-0.022)	LOGIT 0.599 ±.037 (0.077)	RFC 0.565 ±.04 (0.043)	LOGIT 0.576 ±.015 (0.054)	SVM 0.594 ±.026 (0.072)	LOGIT 0.555 ±.025 (0.033)	LOGIT 0.543 ±.022 (0.021)
haberman	DTC 0.401 ±.097	RFC 0.455 ±.108 (0.054)	MLP 0.514 ±.031 (0.113)	MLP 0.509 ±.066 (0.108)	**MLP** **0.515** **±.073** **(0.114)**	LOGIT 0.388 ±.032 (-0.013)	MLP 0.514 ±.032 (0.113)	RFC 0.463 ±.033 (0.062)	RFC 0.426 ±.0 (0.025)	MLP 0.489 ±.003 (0.088)	RFC 0.512 ±.065 (0.111)	MLP 0.471 ±.068 (0.07)
adult	RFC 0.677 ±.001	LOGIT 0.679 ±.005 (0.002)	LOGIT 0.688 ±.001 (0.011)	RFC 0.678 ±.002 (0.001)	**LOGIT** **0.693** **±.003** **(0.016)**	RFC 0.68 ±.003 (0.003)	LOGIT 0.688 ±.001 (0.011)	LOGIT 0.673 ±.011 (-0.004)	RFC 0.674 ±.005 (-0.003)	LOGIT 0.681 ±.003 (0.004)	LOGIT 0.682 ±.005 (0.005)	RFC 0.678 ±.002 (0.001)
yeast3	MLP 0.783 ±.037	**RFC** **0.786** **±.014** **(0.003)**	RFC 0.776 ±.053 (-0.007)	DTC 0.717 ±.018 (-0.066)	RFC 0.76 ±.044 (-0.023)	RFC 0.77 ±.03 (-0.013)	RFC 0.781 ±.02 (-0.002)	**RFC** **0.786** **±.022** **(0.003)**	RFC 0.775 ±.024 (-0.008)	RFC 0.763 ±.024 (-0.02)	RFC 0.751 ±.035 (-0.032)	MLP 0.759 ±.051 (-0.024)
abalone9-18	MLP 0.559 ±.114	MLP 0.508 ±.126 (-0.051)	SVM 0.552 ±.054 (-0.007)	SVM 0.493 ±.074 (-0.066)	MLP 0.539 ±.039 (-0.02)	nan	SVM 0.524 ±.042 (-0.035)	MLP 0.55 ±.086 (-0.009)	SVM 0.493 ±.054 (-0.066)	SVM 0.241 ±.01 (-0.318)	MLP 0.497 ±.07 (-0.062)	**SVM** **0.57** **±.057** **(0.011)**
winequality-red-4	DTC 0.097 ±.032	RFC 0.21 ±.029 (0.113)	MLP 0.184 ±.015 (0.087)	SVM 0.228 ±.034 (0.131)	LOGIT 0.178 ±.01 (0.081)	**LOGIT** **0.236** **±.052** **(0.139)**	KNN 0.21 ±.053 (0.113)	KNN 0.197 ±.083 (0.1)	SVM 0.216 ±.005 (0.119)	DTC 0.145 ±.021 (0.048)	KNN 0.234 ±.064 (0.137)	SVM 0.129 ±.007 (0.032)
mammography	**MLP** **0.69** **±.011**	RFC 0.637 ±.052 (-0.053)	RFC 0.664 ±.015 (-0.026)	RFC 0.522 ±.022 (-0.168)	RFC 0.596 ±.025 (-0.094)	MLP 0.68 ±.019 (-0.01)	RFC 0.683 ±.003 (-0.007)	RFC 0.43 ±.055 (-0.26)	RFC 0.61 ±.08 (-0.08)	RFC 0.66 ±.009 (-0.03)	RFC 0.502 ±.043 (-0.188)	RFC 0.674 ±.023 (-0.016)
abalone-20-vs-8-9-10	MLP 0.414 ±.094	**MLP** **0.46** **±.111** **(0.046)**	MLP 0.378 ±.016 (-0.036)	MLP 0.452 ±.041 (0.038)	MLP 0.374 ±.019 (-0.04)	LOGIT 0.417 ±.0 (0.003)	MLP 0.406 ±.051 (-0.008)	MLP 0.354 ±.06 (-0.06)	DTC 0.326 ±.135 (-0.088)	DTC 0.107 ±.023 (-0.307)	MLP 0.251 ±.072 (-0.163)	MLP 0.454 ±.041 (0.04)

Table 6.1 F1-score without Classifiers HPs Tuning: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

dataset	none	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
pima	**MLP 0.773 ±.029**	MLP 0.749 ±.043 (-0.024)	SVM 0.768 ±.033 (-0.005)	RFC 0.755 ±.048 (-0.018)	MLP 0.771 ±.036 (-0.002)	MLP 0.759 ±.036 (-0.014)	RFC 0.763 ±.033 (-0.01)	RFC 0.742 ±.027 (-0.031)	LOGIT 0.763 ±.036 (-0.01)	MLP 0.752 ±.029 (-0.021)	RFC 0.763 ±.043 (-0.01)	**MLP 0.773 ±.035 (0.0)**
german	RFC 0.757 ±.007	RFC 0.743 ±.008 (-0.014)	RFC 0.751 ±.006 (-0.006)	RFC 0.754 ±.013 (-0.003)	RFC 0.759 ±.01 (0.002)	RFC 0.753 ±.0 (-0.004)	**RFC 0.761 ±.006 (0.004)**	RFC 0.714 ±.019 (-0.043)	RFC 0.747 ±.022 (-0.01)	RFC 0.751 ±.013 (-0.006)	RFC 0.731 ±.011 (-0.026)	RFC 0.758 ±.009 (0.001)
haberman	**SVM 0.739 ±.02**	MLP 0.72 ±.022 (-0.019)	MLP 0.701 ±.049 (-0.038)	MLP 0.697 ±.069 (-0.042)	MLP 0.732 ±.047 (-0.007)	SVM 0.724 ±.034 (-0.015)	MLP 0.716 ±.038 (-0.023)	MLP 0.693 ±.033 (-0.046)	MLP 0.609 ±.0 (-0.13)	SVM 0.67 ±.069 (-0.069)	MLP 0.682 ±.029 (-0.057)	MLP 0.72 ±.053 (-0.019)
adult	RFC 0.85 ±.001	RFC 0.844 ±.002 (-0.006)	RFC 0.837 ±.001 (-0.013)	RFC 0.837 ±.001 (-0.013)	RFC 0.847 ±.001 (-0.003)	RFC 0.849 ±.002 (-0.001)	RFC 0.839 ±.0 (-0.011)	RFC 0.84 ±.002 (-0.01)	RFC 0.835 ±.004 (-0.015)	RFC 0.848 ±.0 (-0.002)	RFC 0.824 ±.005 (-0.026)	**RFC 0.851 ±.001 (0.001)**
yeast3	**MLP 0.953 ±.006**	RFC 0.952 ±.003 (-0.001)	RFC 0.947 ±.012 (-0.006)	DTC 0.931 ±.004 (-0.022)	MLP 0.942 ±.004 (-0.011)	RFC 0.946 ±.007 (-0.007)	RFC 0.949 ±.008 (-0.004)	RFC 0.952 ±.005 (-0.001)	RFC 0.95 ±.004 (-0.003)	RFC 0.95 ±.003 (-0.003)	RFC 0.944 ±.01 (-0.009)	MLP 0.948 ±.008 (-0.005)
abalone9-18	**LOGIT 0.964 ±.006**	MLP 0.942 ±.019 (-0.022)	MLP 0.92 ±.011 (-0.044)	RFC 0.927 ±.026 (-0.037)	MLP 0.927 ±.011 (-0.037)	nan	RFC 0.944 ±.006 (-0.02)	MLP 0.952 ±.013 (-0.012)	LOGIT 0.924 ±.017 (-0.04)	RFC 0.909 ±.006 (-0.055)	MLP 0.911 ±.021 (-0.053)	MLP 0.959 ±.006 (-0.005)
winequality-red-4	**KNN 0.962 ±.003**	KNN 0.927 ±.006 (-0.035)	RFC 0.929 ±.015 (-0.033)	RFC 0.922 ±.016 (-0.04)	RFC 0.881 ±.015 (-0.081)	RFC 0.954 ±.008 (-0.008)	RFC 0.956 ±.004 (-0.006)	KNN 0.948 ±.008 (-0.014)	RFC 0.922 ±.007 (-0.04)	RFC 0.958 ±.003 (-0.004)	KNN 0.891 ±.031 (-0.071)	RFC 0.96 ±.001 (-0.002)
mammography	**MLP 0.983 ±.001**	RFC 0.972 ±.007 (-0.011)	RFC 0.975 ±.001 (-0.008)	RFC 0.953 ±.005 (-0.03)	RFC 0.965 ±.004 (-0.018)	RFC 0.981 ±.002 (-0.002)	RFC 0.982 ±.001 (-0.001)	RFC 0.93 ±.016 (-0.053)	RFC 0.969 ±.009 (-0.014)	RFC 0.98 ±.0 (-0.003)	RFC 0.953 ±.009 (-0.03)	RFC 0.982 ±.001 (-0.001)
abalone-20-vs-8-9-10	MLP 0.988 ±.002	MLP 0.984 ±.006 (-0.004)	RFC 0.977 ±.004 (-0.011)	RFC 0.98 ±.001 (-0.008)	MLP 0.974 ±.004 (-0.014)	RFC 0.988 ±.0 (0.0)	RFC 0.987 ±.001 (-0.001)	KNN 0.977 ±.004 (-0.011)	RFC 0.979 ±.006 (-0.009)	RFC 0.976 ±.001 (-0.012)	KNN 0.945 ±.008 (-0.043)	**MLP 0.989 ±.001 (0.001)**

Table 6.2 Accuracy without Classifiers HPs Tuning: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

results	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
better	6	5	6	5	4	5	5	4	5	5	7
same	0	0	0	0	0	0	0	0	0	0	0
worse	3	4	3	4	4	4	4	5	4	4	2
avg diff [%]	1.8	2.59	1.04	1.41	1.16	3.14	-1.31	-0.3	-4.96	-1.42	1.76
avg std	0.008	-0.022	-0.013	-0.017	-0.02	-0.018	-0.001	-0.01	-0.029	-0.002	-0.014

Table 6.3 F1-score without Classifiers HPs Tuning Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse f1-score than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

From the obtained values, we computed how many times each method has performed better, the same, or worse than using original imbalanced data for the classification[1]. The results for the f1-score can be seen in Table 6.3.

There we can see that none of the oversampling techniques has improved the average f1-score over 3 runs for all datasets. This result supports the fact that dealing with the IDS problem is not an easy task. The maximum number of improvements is for 7 out of 9 datasets, and it was achieved by SMOTified-GAN. LIT-GAN and Borderline-SMOTE improved the classification result in 6 cases considering the f1-score. SMOTE, Poly, LoRAS, CTGAN, TAEI, and CTAB-GAN improved the classification results for 5 datasets. TVAE and k-means-SMOTE were better in 4 cases.

The second part of the table shows the average differences of means and standard deviations from no oversampling over all datasets. A better average of mean differences for the f1-score over all datasets was the best for LoRAS, followed by SMOTE, our method LIT-GAN, SMOTified-GAN, Poly, k-means-SMOTE, and Borderline-SMOTE. The rest of the methods resulted in a worse average difference from no oversampling.

Looking at the standard deviations, only LIT-GAN has a positive average value of the differences between its standard deviation and standard deviation of no oversampling. This means that its results vary the most.

In Table 6.1, we can also see that RFC and MLP classification models gave the best results across all datasets in 31 cases each, followed by LOGIT and SVM, which gave the best results in 21 and 15 cases respectively.

Comparing only LIT-GAN, SMOTified-GAN and TAEI, as our method combines ideas of the latter two, the ratio for being better, the same, or worse than classification with an original dataset for the f1-score, SMOTified-GAN gave the best result 7-0-2. Our method is worse for one more dataset as its result is 6-0-3, and TAEI is worse for one more dataset. However, LIT-GAN has the highest average difference of means from no oversampling from those 3 techniques. Considering only this value, TAEI actually gives the worst result out of all the methods.

Table 6.4 shows the results summary for the accuracy as the evaluation metric. As expected, the accuracy usually lowers after oversampling because it is not a suitable metric for classification on imbalanced datasets, as the classification models can get biased towards the majority class. However, there are oversampling techniques that improve the average accuracy for at least one dataset. Those are Poly, LoRAS, and SMOTified-GAN, where SMOTified-GAN gave higher accuracy 3 times. What is more, SMOTified-GAN and k-means-SMOTE had the same average accuracy as the classification without oversampling in one case. Other methods resulted in worse average accuracy than results obtained on the original imbalanced dataset.

Looking at the average of differences between accuracies obtained on oversampled datasets and original ones, SMOTified-GAN gives the best results, followed by k-means-SMOTE, Lo-

[1]For shortening, in this chapter we use only the names of the oversampling techniques for the results meaning the results of a classification obtained using a dataset oversampled with a given technique. The classification results obtained on the original dataset we denote as no oversampling.

results	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
better	0	0	0	1	0	1	0	0	0	0	3
same	0	0	0	0	1	0	0	0	0	0	1
worse	9	9	9	8	7	8	9	9	9	9	5
avg diff [%]	-1.51	-1.82	-2.37	-1.9	-0.64	-0.8	-2.46	-3.01	-1.94	-3.61	-0.32
avg std	0.005	0.006	0.012	0.006	0.003	0.002	0.006	0.003	0.005	0.01	0.004

Table 6.4 Accuracy without Classifiers HPs Tuning Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse accuracy value than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

metric	none	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
balanced accuracy	0	1	3	2	3	0	1	0	0	0	0	0
accuracy	6	0	0	0	0	0	1	0	0	0	0	3
f1-score	1	2	0	1	3	1	0	1	0	0	0	1
auc	0	1	3	2	3	0	1	0	0	0	0	0
precision	7	0	0	0	0	0	0	0	0	0	0	2
recall	0	1	1	4	1	0	1	0	0	1	2	0
total	14	5	7	9	10	1	4	1	0	1	2	6

Table 6.5 Best Results without Classifiers HPs Tuning: This table shows the number of times each oversampling technique (or no oversampling) gave the best score for a given metric.

RAS, LIT-GAN, SMOTE, Poly, TAEI, Borderline-SMOTE, CTGAN, TVAE, and CTAB-GAN. However, all those values are negative, meaning that the accuracy lowers on average. The average difference of standard deviations is the biggest for Borderline-SMOTE. However, this value is positive for all techniques, meaning that the standard deviations are higher on average than the standard deviations obtained on classification with the original dataset. The best results from Table 6.2 that were summarized in Table 6.4 were achieved mainly by RFC.

Table 6.5 shows the number of times each method gave the best average results over 3 runs with different random seeds. It summarises highlighted values from Tables 6.1 and 6.2 for the f1-score and accuracy and values from Tables in Appendix B for all other evaluation metrics.

Classification without oversampling, considering the f1-score as an evaluation metric, gave the best results for *mammography* dataset. This dataset has the second highest imbalance ratio, concretely 29.902. Our method LIT-GAN performed the best in average for *yeast3* dataset and *abalone-20-vs-8-9-10*. The second dataset has the highest imbalanced ratio of all used datasets, and its value is 72.692. The biggest number of the best average f1-score has Poly oversampling technique, giving the best results for 3 datasets. Considering the accuracy as an evaluation metric, classification without oversampling gives the best result 6 times, followed by SMOTified-GAN with 3 wins and LoRAS with 1 best score.

Focusing only on the combined oversampling techniques, our method is the best considering the f1-score, whereas SMOTified-GAN is the winner for most datasets for the accuracy.

When we look at BA, Poly and SMOTE achieved the best results 3 times. Focusing on AUC, those two techniques again dominate the number of wins. Borderline-SMOTE is the winner in most of the cases considering recall. Recall shows how many times the positive samples were classified correctly. We can see that the obtained results were always better than no oversampling for at least one of the oversampling methods for all datasets. On the other hand, looking at the best precision, no oversampling dominates on most datasets; only SMOTified-GAN is better for 2 out of 9 datasets. This means that for oversampled datasets, more false positives appear. In the total number of wins across all measured metrics, Poly gives the highest number from the oversampling techniques, followed by Borderline-SMOTE, SMOTE, SMOTified-GAN, and our method LIT-GAN in the fifth place.

6.1.2 Results With Tuned Classifiers Hyperparameters

We have run the classification on oversampled datasets for all techniques also 3 times with classifiers' HPs tuning. The tuning was done with grid search using 10-fold stratified cross-validation on the original or balanced training dataset. Some of the classification models, concretely SVM, KNN, DTC, and MLP, are also used in another article comparing traditional oversampling methods [6] with HPs tuning. Inspired by this article, we use the same HPs value options for those classifiers within the grid search. The choices of parameters can be found in Appendix C. We use the same oversampled data as in the case of classification without tuning the hyperparameters instead of rerunning the whole oversampling procedure. The best average results over the 3 runs can be seen in Tables 6.6 and 6.7 for the f1-score and accuracy, and their summaries can be seen in Tables 6.8 and 6.9. Results for other evaluation metrics can be found in Appendix B.

When the HPs of classifiers are tuned, the best performance lowers for 1 out of 9 datasets than in the case of no HPs tuning for the f1-score. Looking at the average differences of the f1-score in Table 6.8, the results indicate that the differences became smaller or even worse for most of the methods in favour of no oversampling. The same holds for the standard deviation differences for most of the methods. There, the average differences of standard deviations from no oversampling are bigger for LIT-GAN, SMOTE, Poly, LoRAS, CTGAN, and SMOTified-GAN.

We can see that in the case of the f1-score, LIT-GAN has 1 same score as no oversampling and 5 higher scores. The ratios of winner-same-worse are the same or worse than in the case of no HPs tuning, except for k-means-SMOTE and LoRAS. LoRAS is better for 7 datasets, which is the highest number. Other methods achieved better results than no oversampling for 5 datasets at most.

In Table 6.6, RFC, followed by MLP and LOGIT, gave the best results in most cases.

Considering only LIT-GAN, SMOTified-GAN, and TAEI, our method has the best ratio of being better-same-worse than no oversampling. However, SMOTified-GAN, has higher average differences of means for the f1-score.

Looking at the accuracy, most of the classifications with oversampling obtained worse results than those without oversampling. RFC is the classification model that gave the best results considering the accuracy in more than half of the cases. The average of differences from no oversampling in Table 6.9 is the best for SMOTified-GAN, followed by k-means-SMOTE, LoRAS, and our method in the fourth place. SMOTified-GAN and LoRAS achieved higher accuracy for 2 datasets than no oversampling. SMOTE, Poly, k-means-SMOTE and TAEI were better for 1 dataset. The other techniques performed worse than the classification done on the original dataset for all datasets considering the accuracy.

Comparing only combined oversampling techniques LIT-GAN, TAEI, ald SMOTified-GAN, our method LIT-GAN gave the worst ratio of being better-same-worse than no oversampling considering the accuracy. However, the average differences of means and the average difference of standard deviations from no oversampling are higher, respectively lower, for LIT-GAN than for TAEI.

dataset	none	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
pima	RFC 0.636 ±.028	SVM 0.644 ±.045 (0.008)	LOGIT 0.684 ±.046 (0.048)	**LOGIT** **0.687** **±.052** **(0.051)**	RFC 0.683 ±.036 (0.047)	RFC 0.668 ±.031 (0.032)	SVM 0.68 ±.043 (0.044)	SVM 0.666 ±.043 (0.03)	LOGIT 0.678 ±.033 (0.042)	LOGIT 0.673 ±.058 (0.037)	MLP 0.669 ±.018 (0.033)	LOGIT 0.678 ±.049 (0.042)
german	MLP 0.562 ±.02	SVM 0.571 ±.025 (0.009)	SVM 0.589 ±.023 (0.027)	**SVM** **0.612** **±.017** **(0.05)**	LOGIT 0.581 ±.033 (0.019)	LOGIT 0.497 ±.0 (-0.065)	LOGIT 0.599 ±.037 (0.037)	RFC 0.576 ±.036 (0.014)	LOGIT 0.576 ±.015 (0.014)	SVM 0.592 ±.017 (0.03)	MLP 0.552 ±.033 (-0.01)	LOGIT 0.54 ±.02 (-0.022)
haberman	DTC 0.391 ±.075	DTC 0.457 ±.006 (0.066)	MLP 0.476 ±.077 (0.085)	MLP 0.488 ±.079 (0.097)	MLP 0.501 ±.092 (0.11)	LOGIT 0.405 ±.025 (0.014)	MLP 0.457 ±.06 (0.066)	RFC 0.475 ±.007 (0.084)	DTC 0.426 ±.0 (0.035)	RFC 0.501 ±.043 (0.11)	**RFC** **0.527** **±.044** **(0.136)**	MLP 0.477 ±.061 (0.086)
adult	RFC 0.679 ±.002	LOGIT 0.679 ±.005 (0.0)	LOGIT 0.688 ±.001 (0.009)	RFC 0.68 ±.004 (0.001)	**LOGIT** **0.693** **±.003** **(0.014)**	RFC 0.681 ±.004 (0.002)	LOGIT 0.688 ±.001 (0.009)	RFC 0.674 ±.005 (-0.005)	RFC 0.675 ±.006 (-0.004)	LOGIT 0.682 ±.002 (0.003)	LOGIT 0.682 ±.005 (0.003)	LOGIT 0.679 ±.008 (0.0)
yeast3	MLP 0.783 ±.037	RFC 0.778 ±.018 (-0.005)	RFC 0.765 ±.038 (-0.018)	RFC 0.731 ±.033 (-0.052)	RFC 0.758 ±.04 (-0.025)	RFC 0.767 ±.028 (-0.016)	**RFC** **0.787** **±.027** **(0.004)**	RFC 0.784 ±.022 (0.001)	RFC 0.759 ±.024 (-0.024)	DTC 0.772 ±.031 (-0.011)	DTC 0.786 ±.029 (0.003)	DTC 0.768 ±.041 (-0.015)
abalone9-18	**LOGIT** **0.618** **±.079**	MLP 0.501 ±.123 (-0.117)	MLP 0.543 ±.06 (-0.075)	SVM 0.517 ±.07 (-0.101)	MLP 0.555 ±.065 (-0.063)	nan	MLP 0.579 ±.067 (-0.039)	MLP 0.515 ±.109 (-0.103)	SVM 0.493 ±.054 (-0.125)	LOGIT 0.236 ±.015 (-0.382)	MLP 0.513 ±.047 (-0.105)	LOGIT 0.597 ±.087 (-0.021)
winequality-red-4	KNN 0.122 ±.043	RFC 0.219 ±.066 (0.097)	MLP 0.193 ±.05 (0.071)	LOGIT 0.229 ±.036 (0.107)	LOGIT 0.177 ±.009 (0.055)	**LOGIT** **0.239** **±.054** **(0.117)**	KNN 0.22 ±.044 (0.098)	SVM 0.193 ±.03 (0.071)	KNN 0.235 ±.049 (0.113)	LOGIT 0.132 ±.005 (0.01)	**KNN** **0.239** **±.062** **(0.117)**	LOGIT 0.128 ±.014 (0.006)
mammography	MLP 0.691 ±.021	RFC 0.648 ±.05 (-0.043)	RFC 0.665 ±.005 (-0.026)	RFC 0.521 ±.012 (-0.17)	RFC 0.61 ±.022 (-0.081)	**MLP** **0.697** **±.019** **(0.006)**	RFC 0.692 ±.011 (0.001)	RFC 0.436 ±.047 (-0.255)	RFC 0.61 ±.08 (-0.081)	RFC 0.663 ±.027 (-0.028)	RFC 0.497 ±.044 (-0.194)	RFC 0.674 ±.023 (-0.017)
abalone-20-vs-8-9-10	LOGIT 0.421 ±.058	MLP 0.442 ±.091 (0.021)	MLP 0.387 ±.073 (-0.034)	MLP 0.369 ±.055 (-0.052)	MLP 0.389 ±.088 (-0.032)	SVM 0.4 ±.0 (-0.021)	MLP 0.414 ±.092 (-0.007)	MLP 0.351 ±.073 (-0.07)	MLP 0.288 ±.054 (-0.133)	MLP 0.157 ±.061 (-0.264)	MLP 0.252 ±.071 (-0.169)	**MLP** **0.478** **±.071** **(0.057)**

Table 6.6 F1-score with Classifiers HPs Tuning: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

dataset	none	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
pima	RFC 0.763 ±.032	MLP 0.745 ±.028 (-0.018)	**RFC** **0.769** **±.037** **(0.006)**	RFC 0.756 ±.047 (-0.007)	MLP 0.763 ±.039 (0.0)	RFC 0.768 ±.033 (0.005)	RFC 0.766 ±.037 (0.003)	RFC 0.743 ±.023 (-0.02)	LOGIT 0.763 ±.036 (0.0)	LOGIT 0.743 ±.05 (-0.02)	RFC 0.749 ±.037 (-0.014)	MLP 0.768 ±.039 (0.005)
german	RFC 0.754 ±.007	RFC 0.726 ±.026 (-0.028)	RFC 0.743 ±.012 (-0.011)	RFC 0.753 ±.007 (-0.001)	RFC 0.757 ±.014 (0.003)	RFC 0.747 ±.0 (-0.007)	RFC 0.756 ±.01 (0.002)	RFC 0.719 ±.019 (-0.035)	MLP 0.73 ±.031 (-0.024)	RFC 0.758 ±.007 (0.004)	RFC 0.732 ±.01 (-0.022)	**RFC** **0.759** **±.008** **(0.005)**
haberman	**SVM** **0.739** **±.02**	MLP 0.728 ±.029 (-0.011)	MLP 0.678 ±.074 (-0.061)	MLP 0.674 ±.087 (-0.065)	MLP 0.724 ±.056 (-0.015)	SVM 0.732 ±.033 (-0.007)	MLP 0.67 ±.062 (-0.069)	RFC 0.67 ±.057 (-0.069)	DTC 0.598 ±.0 (-0.141)	SVM 0.67 ±.069 (-0.069)	MLP 0.69 ±.038 (-0.049)	MLP 0.728 ±.046 (-0.011)
adult	**RFC** **0.852** **±.001**	RFC 0.845 ±.004 (-0.007)	RFC 0.843 ±.007 (-0.009)	RFC 0.844 ±.007 (-0.008)	RFC 0.849 ±.003 (-0.003)	RFC 0.851 ±.003 (-0.001)	RFC 0.839 ±.0 (-0.013)	RFC 0.842 ±.004 (-0.01)	RFC 0.836 ±.006 (-0.016)	RFC 0.848 ±.0 (-0.004)	RFC 0.825 ±.006 (-0.027)	RFC 0.851 ±.001 (-0.001)
yeast3	**MLP** **0.953** **±.006**	RFC 0.95 ±.003 (-0.003)	RFC 0.943 ±.01 (-0.01)	RFC 0.93 ±.02 (-0.023)	RFC 0.942 ±.014 (-0.011)	RFC 0.945 ±.006 (-0.008)	RFC 0.95 ±.009 (-0.003)	RFC 0.952 ±.005 (-0.001)	RFC 0.947 ±.0 (-0.006)	RFC 0.949 ±.004 (-0.004)	DTC 0.948 ±.008 (-0.005)	RFC 0.95 ±.004 (-0.003)
abalone9-18	**MLP** **0.962** **±.002**	MLP 0.938 ±.018 (-0.024)	MLP 0.93 ±.013 (-0.032)	RFC 0.924 ±.024 (-0.038)	MLP 0.935 ±.012 (-0.027)	nan	RFC 0.942 ±.004 (-0.02)	MLP 0.947 ±.015 (-0.015)	MLP 0.924 ±.02 (-0.038)	RFC 0.909 ±.006 (-0.053)	MLP 0.92 ±.014 (-0.042)	LOGIT 0.961 ±.006 (-0.001)
winequality-red-4	**SVM** **0.961** **±.0**	RFC 0.927 ±.016 (-0.034)	RFC 0.922 ±.019 (-0.039)	RFC 0.924 ±.018 (-0.037)	MLP 0.894 ±.022 (-0.067)	RFC 0.958 ±.003 (-0.003)	RFC 0.953 ±.004 (-0.008)	KNN 0.944 ±.0 (-0.017)	RFC 0.922 ±.008 (-0.039)	RFC 0.958 ±.003 (-0.003)	KNN 0.894 ±.031 (-0.067)	**RFC** **0.961** **±.0** **(0.0)**
mammography	**MLP** **0.983** **±.001**	RFC 0.973 ±.006 (-0.01)	RFC 0.975 ±.0 (-0.008)	RFC 0.953 ±.004 (-0.03)	RFC 0.967 ±.004 (-0.016)	RFC 0.981 ±.002 (-0.002)	RFC 0.982 ±.001 (-0.001)	RFC 0.933 ±.013 (-0.05)	RFC 0.969 ±.009 (-0.014)	RFC 0.98 ±.002 (-0.003)	RFC 0.954 ±.009 (-0.029)	RFC 0.982 ±.001 (-0.001)
abalone-20-vs-8-9-10	**LOGIT** **0.989** **±.001**	MLP 0.983 ±.005 (-0.006)	MLP 0.978 ±.003 (-0.011)	RFC 0.98 ±.001 (-0.009)	MLP 0.978 ±.004 (-0.011)	RFC 0.986 ±.0 (-0.003)	RFC 0.987 ±.001 (-0.002)	KNN 0.976 ±.001 (-0.013)	RFC 0.977 ±.007 (-0.012)	RFC 0.976 ±.002 (-0.013)	MLP 0.941 ±.014 (-0.048)	**MLP** **0.989** **±.001** **(0.0)**

Table 6.7 Accuracy with Classifiers HPs tuning: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

results	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
better	5	5	5	5	5	7	5	4	5	5	4
same	1	0	0	0	0	0	0	0	0	0	1
worse	3	4	4	4	3	2	4	5	4	4	4
avg diff [%]	0.4	0.97	-0.77	0.49	0.86	2.37	-2.59	-1.81	-5.5	-2.07	1.29
avg std	0.007	0.001	-0.001	0.003	-0.015	0.002	0.001	-0.005	-0.012	-0.001	0.001

Table 6.8 F1-score with Classifiers HPs Tuning Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse f1-score than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

results	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
better	0	1	0	1	1	2	0	0	1	0	2
same	0	0	0	1	0	0	0	1	0	0	2
worse	9	8	9	7	7	7	9	8	8	9	5
avg diff [%]	-1.57	-1.94	-2.42	-1.63	-0.32	-1.23	-2.56	-3.22	-1.83	-3.37	-0.08
avg std	0.007	0.012	0.016	0.011	0.002	0.006	0.007	0.005	0.008	0.011	0.004

Table 6.9 Accuracy with Classifiers HPs Tuning Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse accuracy value than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

The winning times for each method and evaluation metric can be seen in Table 6.10 for all measured metrics. Looking at those metrics, our method is the best for 1 dataset in the case of BA, AUC, and recall. Compared to the results without classifiers' HPs tuning, where Poly gave the best results, here we can see that from the oversampling techniques, Borderline-SMOTE performs the best, considering the number of being the winner across all metrics. The numbers for classification without oversampling are the same compared to the classification without HPs tuning, except for the accuracy. When the classifiers' hyperparameters are tuned, no oversampling is the best for one more dataset for the accuracy as the evaluation metric.

metric	none	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
balanced accuracy	0	1	1	3	1	0	2	0	0	0	1	0
accuracy	7	0	1	0	0	0	0	0	0	0	0	3
f1-score	1	0	0	2	1	2	1	0	0	0	2	1
auc	0	1	1	3	1	0	2	0	0	0	1	0
precision	7	0	0	0	0	0	0	0	0	0	0	2
recall	0	1	1	5	0	0	1	0	0	1	1	0
total	15	3	4	13	3	2	6	0	0	1	5	6

Table 6.10 Best Results with Classifiers HPs Tuning: This table shows the number of times each oversampling technique (or no oversampling) gave the best score for a given metric.

6.2 Change of Interpolation Method

As mentioned in Chapter 5, LIT-GAN can be used together with any interpolation method, such as SMOTE. For a comparison with TAEI and SMOTified-GAN, we used SMOTE in the previous experiments. In this Section, we compare LIT-GAN performance with different interpolation methods as its oversampler, concretely SMOTE, Poly, and Borderline-SMOTE. We use Poly with the Mesh topology, and from the Borderline-SMOTE variants, we use Borderline-SMOTE2.

The experiment was done as in Section 6.1.1. The hyperparameters of classifiers were not tuned, and the program ran 3 times with different random seeds. The resulting average f1-score and accuracy of those 3 runs can be seen in Tables 6.11 and 6.13. A summary of the comparison of LIT-GAN variants and the classification done on the original dataset can be seen in Tables 6.12 and 6.14. For the rest of the evaluation metrics, the results can be found in Appendix B.

dataset	none	litgan-smote	litgan-poly	litgan-borderline
pima	MLP 0.658 ±.028	MLP 0.657 ±.042 (-0.001)	**LOGIT 0.668 ±.045 (0.01)**	MLP 0.659 ±.021 (0.001)
german	MLP 0.522 ±.027	**LOGIT 0.571 ±.027 (0.049)**	SVM 0.569 ±.028 (0.047)	SVM 0.57 ±.041 (0.048)
haberman	DTC 0.401 ±.097	**RFC 0.455 ±.108 (0.054)**	RFC 0.375 ±.092 (-0.026)	RFC 0.409 ±.072 (0.008)
adult	RFC 0.677 ±.001	LOGIT 0.679 ±.005 (0.002)	LOGIT 0.675 ±.005 (-0.002)	**LOGIT 0.68 ±.003 (0.003)**
yeast3	MLP 0.783 ±.037	**RFC 0.786 ±.014 (0.003)**	RFC 0.781 ±.016 (-0.002)	RFC 0.775 ±.029 (-0.008)
abalone9-18	MLP 0.559 ±.114	MLP 0.508 ±.126 (-0.051)	RFC 0.465 ±.058 (-0.094)	**MLP 0.575 ±.1 (0.016)**
winequality-red-4	DTC 0.097 ±.032	RFC 0.21 ±.029 (0.113)	MLP 0.233 ±.05 (0.136)	**SVM 0.235 ±.052 (0.138)**
mammography	**MLP 0.69 ±.011**	RFC 0.637 ±.052 (-0.053)	RFC 0.606 ±.021 (-0.084)	RFC 0.649 ±.043 (-0.041)
abalone-20-vs-8-9-10	MLP 0.414 ±.094	**MLP 0.46 ±.111 (0.046)**	MLP 0.335 ±.105 (-0.079)	MLP 0.399 ±.072 (-0.015)

Table 6.11 F1-score of LIT-GAN Variants: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

Based on Table 6.11 and 6.12, we can see that none of the interpolation techniques used within LIT-GAN achieved a higher f1-score on the *mammography* dataset, which is a dataset with the second highest IR from those used in our work. The average difference between the f1-score measured on the oversampled dataset and the f1-score of classification without oversampling of the training dataset is positive for LIT-GAN + SMOTE and LIT-GAN + Borderline-SMOTE interpolation methods, with the higher value for the first mentioned.

If we look at the number of times each oversampling method gave a higher f1-score than no oversampling, we can see that again LIT-GAN + SMOTE and LIT-GAN + Borderline-SMOTE resulted in having the same result, concretely being better for 6 datasets out of 9, LIT-GAN + Poly performed better for less than a half of the datasets, being better only on *pima*, *german*, and

results	litgan-smote	litgan-poly	litgan-borderline
better	6	3	6
same	0	0	0
worse	3	6	3
avg diff [%]	1.8	-1.04	1.67
avg std	0.008	-0.002	-0.001

Table 6.12 F1-score of LIT-GAN Variants Summary: Table shows how many times classification with the oversampled dataset for each variant gives better, the same, or worse f1-score than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

winequality-red-4 datasets.

dataset	none	litgan-smote	litgan-poly	litgan-borderline
pima	**MLP 0.773 ±.029**	MLP 0.749 ±.043 (-0.024)	MLP 0.762 ±.032 (-0.011)	MLP 0.76 ±.026 (-0.013)
german	**RFC 0.757 ±.007**	RFC 0.743 ±.008 (-0.014)	RFC 0.749 ±.018 (-0.008)	RFC 0.746 ±.009 (-0.011)
haberman	**SVM 0.739 ±.02**	MLP 0.72 ±.022 (-0.019)	RFC 0.617 ±.096 (-0.122)	MLP 0.674 ±.053 (-0.065)
adult	**RFC 0.85 ±.001**	RFC 0.844 ±.002 (-0.006)	RFC 0.845 ±.001 (-0.005)	RFC 0.845 ±.003 (-0.005)
yeast3	**MLP 0.953 ±.006**	RFC 0.952 ±.003 (-0.001)	RFC 0.951 ±.002 (-0.002)	RFC 0.95 ±.006 (-0.003)
abalone9-18	**LOGIT 0.964 ±.006**	MLP 0.942 ±.019 (-0.022)	KNN 0.941 ±.004 (-0.023)	MLP 0.959 ±.011 (-0.005)
winequality-red-4	**KNN 0.962 ±.003**	KNN 0.927 ±.006 (-0.035)	MLP 0.902 ±.023 (-0.06)	RFC 0.955 ±.003 (-0.007)
mammography	**MLP 0.983 ±.001**	RFC 0.972 ±.007 (-0.011)	RFC 0.968 ±.003 (-0.015)	RFC 0.975 ±.005 (-0.008)
abalone-20-vs-8-9-10	**MLP 0.988 ±.002**	MLP 0.984 ±.006 (-0.004)	KNN 0.973 ±.011 (-0.015)	**MLP 0.988 ±.001 (0.0)**

Table 6.13 Accuracy of LIT-GAN Variants: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

Looking at the accuracy in Table 6.13, the classification model trained on the original training

results	litgan-smote	litgan-poly	litgan-borderline
better	0	0	0
same	0	0	1
worse	9	9	8
avg diff [%]	-1.51	-2.9	-1.3
avg std	0.005	0.013	0.005

Table 6.14 Accuracy of LIT-GAN Variants Summary: Table shows how many times classification with the oversampled dataset for each variant gives better, the same, or worse accuracy value than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

metric	none	litgan-smote	litgan-poly	litgan-borderline
balanced accuracy	0	7	2	0
accuracy	9	0	0	1
f1-score	1	4	1	3
auc	0	7	2	0
precision	9	0	0	0
recall	0	6	3	0
total	19	24	8	4

Table 6.15 Best Results of LIT-GAN Variants: This table shows the number of times each variant or no oversampling gave the best score for a given metric.

dataset gave the best accuracy value for all 9 datasets. However, this is not surprising, as mentioned in Section 6.1.1. From LIT-GAN variants, only LIT-GAN + Borderline-SMOTE achieved the same accuracy on *abalone-20-vs-8-9-10* dataset.

Table 6.15 shows the number of winning times for each LIT-GAN variant for all measured metrics. LIT-GAN + SMOTE has the highest number of winning times for all metrics, except for the accuracy and precision, where no oversampling dominates. There, we can also see that LIT-GAN + Poly has more wins than LIT-GAN + Borderline-SMOTE focusing on BA, AUC, and recall.

We also measured the duration of oversampling using different interpolation methods. The results are in seconds and can be seen in Table 6.16. The datasets are sorted in ascending order based on their size. The differences are not significant. However, LIT-GAN + SMOTE is the slowest, except for the 2 biggest datasets. On those 2 datasets LIT-GAN + Borderline-SMOTE takes the longest time to oversample.

To summarize, 2 LIT-GAN variants give better results, considering the f1-score for more than half of the datasets, concretely 6 out of 9. Those variants are LIT-GAN + SMOTE and LIT-GAN + Borderline-SMOTE. Considering the average difference of mean values from no oversampling, LIT-GAN + SMOTE would be a better option because it gives a higher average difference for the f1-score. On the other hand, the differences in standard deviations from no oversampling are, on average, smaller for LIT-GAN + Borderline-SMOTE. From the accuracy point of view, LIT-GAN + Borderline-SMOTE is the only variant that achieved at least the same value for one dataset as no oversampling. However, looking at Table 6.15, LIT-GAN + SMOTE dominates from the presented LIT-GAN variants for all metrics, except for the accuracy. Considering this fact, LIT-GAN + SMOTE would be the best choice from the proposed variants. From the duration point of view, for the smaller datasets, LIT-GAN variants with Poly and Borderline-SMOTE are faster, but as the number of samples increases, LIT-GAN +SMOTE again dominates.

dataset	litgan-smote	litgan-poly	litgan-borderline
haberman	115.7614	104.0266	103.7043
abalone9-18	510.9182	428.072	456.4113
pima	311.6711	265.483	257.9664
german	505.3241	420.2579	427.5395
winequality-red-4	1055.5894	859.2533	897.6886
yeast3	454.6102	418.8299	432.1946
abalone-20_vs_8-9-10	671.3989	642.7156	768.59
mammography	1586.2494	1687.5678	1711.9607
adult	4365.0659	5034.1993	5296.1138

Table 6.16 Duration of LIT-GAN Variants in Seconds

6.3 Time

Next to the classification performance, we also measured the duration of oversampling for each technique. The results are in Table 6.17. The times are averaged from the 3 oversampling runs with different random seeds. The datasets are sorted in increasing order based on the number of observations. As we do not have results for the combination of k-means-SMOTE and *abalone9-18* dataset, the duration is not presented.

dataset	lit-gan (ours)	smote	borderline	poly	k_means	loras	ctgan	tvae	taei	ctab_gan	smotified_gan
haberman	115.7614	0.0037	0.0047	0.0012	0.0188	0.84	10.4762	4.4451	12.2716	10.2554	10.9292
abalone9-18	510.9182	0.0047	0.0058	0.002	nan	6.4497	35.0422	23.326	17.6299	52.1691	27.0309
pima	311.6711	0.005	0.0073	0.0013	0.0329	0.9895	37.0067	16.614	22.6001	56.5302	27.7271
german	505.3241	0.0125	0.0164	0.0016	0.4592	2.3454	73.9081	21.2383	34.0925	86.9136	39.9359
winequality-red-4	1055.5894	0.0045	0.0065	0.0025	0.466	9.8999	70.1114	48.916	24.156	104.3212	49.8295
yeast3	454.6102	0.0045	0.0078	0.0022	0.0786	5.081	35.4325	17.6938	35.1945	66.5927	45.8387
abalone-20_vs_8-9-10	671.3989	0.0044	0.0064	0.0045	0.8463	29.4004	41.3515	55.8164	40.4723	89.1225	69.4966
mammography	1586.2494	0.0054	0.0099	0.0079	1.1386	25.0117	125.5668	53.6856	217.3072	143.6011	527.0128
adult	4365.0659	0.1439	0.3459	0.0491	1.4829	85.9357	643.0461	202.9194	1794.6873	1109.6364	4724.0642

Table 6.17 Oversampling Durations in Seconds

We can see that traditional methods are much faster than those using generative models. This is due to the fact that generative-based techniques require training of the networks, whereas the traditional ones do not. LIT-GAN, as it combines two generative networks and one interpolation method, takes the longest time for oversampling, except for *adult* dataset, where SMOTified-GAN takes the longest time. The two networks of LIT-GAN have to be trained sequentially because GAN is dependent on the VAE output. SMOTE, Borderline-SMOTE, and Poly take less than a second to oversample each one of the datasets. k-means-SMOTE takes more than one second to oversample the two largest datasets, *mammography* and *adult*. LoRAS, one of the newest traditional methods, takes more than 85 seconds to oversample for the biggest dataset.

6.4 Discussion

Based on our experiments, traditional oversampling methods are the best choice from the duration point of view. The reason is that GANs and AEs need training, and traditional methods do not, which can be seen as their advantage.

When it comes to the evaluation with the f1-score as the evaluation metric, there is no method that would outperform others on all datasets. If we consider only the number of best performances across all measured metrics, Poly performs the best when classifiers' HPs are not tuned, and Borderline-SMOTE is the best when they are tuned.

Our method, LIT-GAN, gives competitive results to other oversampling techniques. However, as it combines two generative models, its downside is the time consumption.

From the results of our experiment, we can see that traditional oversampling methods also outperform the generative ones when it comes to being better than no oversampling on average on all datasets while using the f1-score and accuracy as the evaluation metrics. From the combined techniques, LIT-GAN and SMOTified-GAN give competitive results considering the f1-score. In the case of experiments with HPs tuning of the classifiers, SMOTified-GAN gives the highest average difference from the classification with no oversampling over all datasets for the accuracy and second highest for the f1-score.

Looking at data preprocessing, generative-based techniques handle the preprocessing of categorical and continuous features within their architectures, which is not the case for traditional methods. In fact, to the best of our knowledge, only SMOTE-NC is a traditional oversampling technique that handles categorical features explicitly. Other techniques need external preprocessing, such as one-hot encoding. This can be seen as a disadvantage of traditional methods compared to the generative-based ones. Focusing on the combined techniques, our implementation handles the data preprocessing in the same way as generative-based methods, while SMOTified-GAN needs external preprocessing. For TAEI, categorical features have to be embedded externally using an `OrdinalEncoder` before they are used as an input of TAEI, and continuous features have to be preprocessed as in the case of traditional techniques.

Chapter 7
Conclusion

The aim of this thesis was to do an extensive survey of the existing oversampling techniques for imbalanced binary classification tabular datasets, select at least five of them, and implement, evaluate and compare those selected techniques. What is more, a new oversampling technique had to be presented, implemented, and compared to the other techniques, and the results should have been discussed.

There exist dozens of methods, especially the traditional ones, together with their overviews like the one presented by Kovács [6]. In Chapter 2, we have outlined commonly used traditional and generative-based methods and their combinations. Chapter 3 contains a more detailed description of the selected 10 methods used in this work for comparison and performance evaluation. Chapter 4 consists of the description of the used metrics, classifiers, and datasets, together with the implementation details of the experiments. In this thesis, we use 6 classification models, 6 evaluation metrics, and 9 datasets. In Chapter 5, our novel oversampling technique, which we called Latent Interpolation Tabular GAN, is presented. Details about the data preprocessing and its architecture, the same as LIT-GAN training and synthetic data generation, are presented there. Chapter 6 shows the performance of the techniques using combinations of different classification models and metrics, durations of oversampling for each combination of oversampling technique and dataset, comparison of the methods, and discussion over the results and advantages and disadvantages of the techniques.

7.1 Contribution

In this thesis, we present a novel oversampling technique for tabular data balancing in the binary classification problem. We call our method Latent Interpolation Tabular GAN (LIT-GAN). It is competitive with other oversampling techniques, considering performance. LIT-GAN can be used together with datasets of both numerical and categorical features. The oversampling approach of LIT-GAN combines the ideas of two other methods, concretely TAEI and SMOTified-GAN, to generate synthetic data matching the minority class distribution. However, we use different preprocessing steps, the architecture of the network, and the training of GAN also slightly differs from SMOTified-GAN.

What is more, in the research part of our work, we present 28 techniques that can be used for oversampling the training dataset. However, articles on those oversampling techniques use different preprocessing steps, experiment setup, and evaluation metrics, making their results incomparable. We have done a comparison of selected traditional, generative-based and combined techniques using 6 classification models and 6 evaluation metrics on 9 datasets frequently used in works that focus on the IDS problem.

7.2 Future Work

Given the results presented in Chapter 6, the novel method LIT-GAN does not systematically outperform SMOTified-GAN, although LIT-GAN adopts SMOTified-GAN idea. However, the preprocessing, architecture, and training processes differ in LIT-GAN. All those steps have an effect on the oversampling quality. Therefore, as future work, different preprocessing can be used for LIT-GAN.

What is more, our method accepts as a parameter oversampler. The oversampler can be any interpolation method such as SMOTE, Borderline-SMOTE, or Poly, as shown in Section 6.2. A comparison of more interpolation techniques within our method could be a possible extension of this thesis.

Furthermore, as we measured the performance on each dataset more times, statistical tests like the Friedman test could be used to compare the oversampling techniques to determine whether classifications on the oversampled datasets perform similarly or not. The same holds for different LIT-GAN variants, such as the ones presented in Section 6.2.

Last but not least, although this thesis concentrates on tabular datasets for binary classification, as a future work, the idea of LIT-GAN could be extended for multi-class dataset classification problems.

Bibliography

1. KLOUDA, Karel. *Strojové učení 1: Supervizované učení, klasifikační úloha, rozhodovací stromy* [online]. 2022. [visited on 2023-09-08]. Available from: `https://courses.fit.cvut.cz/BI-ML1/lectures/files/BI-ML1-02-cs-handout.pdf`. [File available after signing into CTU website - copy of this file is available on the attached CD].
2. FRIEDJUNGOVÁ, Magda. *Předzpracování dat: Úvod, analýza dat, validace a čištění* [online]. 2023. [visited on 2023-09-09]. Available from: `https://courses.fit.cvut.cz/NI-PDD/tutorials/index.html`. [File available after signing into CTU website - copy of this file is available on the attached CD].
3. LEMNARU, Camelia; POTOLEA, Rodica. Imbalanced Classification Problems: Systematic Study, Issues and Best Practices. In: ZHANG, Runtong; ZHANG, Juliang; ZHANG, Zhenji; FILIPE, Joaquim; CORDEIRO, José (eds.). *Enterprise Information Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 35–50. ISBN 978-3-642-29958-2.
4. CHAWLA, Nitesh V. Data Mining for Imbalanced Datasets: An Overview. In: *Data Mining and Knowledge Discovery Handbook*. Ed. by MAIMON, Oded; ROKACH, Lior. Boston, MA: Springer US, 2010, pp. 875–886. ISBN 978-0-387-09823-4. Available from DOI: `10.1007/978-0-387-09823-4_45`.
5. VISA, Sofia; RALESCU, Anca. Issues in mining imbalanced data sets-a review paper. In: *Proceedings of the sixteen midwest artificial intelligence and cognitive science conference*. sn, 2005, vol. 2005, pp. 67–73.
6. KOVÁCS, György. An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets. *Applied Soft Computing*. 2019, vol. 83, p. 105662. ISSN 1568-4946. Available from DOI: `https://doi.org/10.1016/j.asoc.2019.105662`.
7. HE, Haibo; GARCIA, Edwardo A. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*. 2009, vol. 21, no. 9, pp. 1263–1284. Available from DOI: `10.1109/TKDE.2008.239`.
8. CANUMA, Pronce. *How to Deal With Imbalanced Classification and Regression Data* [online]. Neptune AI, 2023 [visited on 2023-09-10]. Available from: `https://neptune.ai/blog/how-to-deal-with-imbalanced-classification-and-regression-data`.
9. KRAWCZYK, Bartosz. Learning from imbalanced data: Open challenges and future directions. *Progress in Artificial Intelligence*. 2016, vol. 5. Available from DOI: `10.1007/s13748-016-0094-0`.
10. GAZZAH, Sami; ESSOUKRI BEN AMARA, Najoua. New Oversampling Approaches Based on Polynomial Fitting for Imbalanced Data Sets. In: 2008, pp. 677–684. Available from DOI: `10.1109/DAS.2008.74`.

11. LING, Charles X.; SHENG, Victor S. Cost-Sensitive Learning. In: *Encyclopedia of Machine Learning*. Ed. by SAMMUT, Claude; WEBB, Geoffrey I. Boston, MA: Springer US, 2010, pp. 231–235. ISBN 978-0-387-30164-8. Available from DOI: 10.1007/978-0-387-30164-8_181.

12. GOODFELLOW, Ian J.; POUGET-ABADIE, Jean; MIRZA, Mehdi; XU, Bing; WARDE-FARLEY, David; OZAIR, Sherjil; COURVILLE, Aaron; BENGIO, Yoshua. *Generative Adversarial Networks*. 2014. Available from arXiv: 1406.2661 [stat.ML].

13. HAN, Hui; WANG, Wen-Yuan; MAO, Bing-Huan. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In: HUANG, De-Shuang; ZHANG, Xiao-Ping; HUANG, Guang-Bin (eds.). *Advances in Intelligent Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 878–887. ISBN 978-3-540-31902-3.

14. CHAWLA, N. V.; BOWYER, K. W.; HALL, L. O.; KEGELMEYER, W. P. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*. 2002, vol. 16, pp. 321–357. Available from DOI: 10.1613/jair.953.

15. BATISTA, Gustavo; PRATI, Ronaldo; MONARD, Maria-Carolina. Balancing Strategies and Class Overlapping. In: 2005, pp. 24–35. ISBN 3-540-28795-7. Available from DOI: 10.1007/11552253_3.

16. FERNÁNDEZ, Alberto; GARCÍA, Salvador; HERRERA, Francisco; CHAWLA, N. SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary. *J. Artif. Intell. Res.* 2018, vol. 61, pp. 863–905. Available also from: https://api.semanticscholar.org/CorpusID:3373087.

17. BARUA, Sukarna; ISLAM, Md. Monirul; MURASE, Kazuyuki. ProWSyn: Proximity Weighted Synthetic Oversampling Technique for Imbalanced Data Set Learning. In: PEI, Jian; TSENG, Vincent S.; CAO, Longbing; MOTODA, Hiroshi; XU, Guandong (eds.). *Advances in Knowledge Discovery and Data Mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 317–328. ISBN 978-3-642-37456-2.

18. BEJ, Saptarshi; DAVTYAN, Narek; WOLFIEN, Markus; NASSAR, Mariam; WOLKENHAUER, Olaf. *LoRAS: An oversampling approach for imbalanced datasets*. 2020. Available from arXiv: 1908.08346 [cs.LG].

19. HE, Haibo; BAI, Yang; GARCIA, Edwardo A.; LI, Shutao. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. 2008, pp. 1322–1328. Available from DOI: 10.1109/IJCNN.2008.4633969.

20. BUNKHUMPORNPAT, Chumphol; SINAPIROMSARAN, Krung; LURSINSAP, Chidchanok. Safe-Level-SMOTE: Safe-Level-Synthetic Minority Over-Sampling TEchnique for Handling the Class Imbalanced Problem. In: THEERAMUNKONG, Thanaruk; KIJSIRIKUL, Boonserm; CERCONE, Nick; HO, Tu-Bao (eds.). *Advances in Knowledge Discovery and Data Mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 475–482. ISBN 978-3-642-01307-2.

21. DANG, Xuan Tho; TRAN, Dang Hung; HIROSE, Osamu; SATOU, Kenji. SPY: A Novel Resampling Method for Improving Classification Performance in Imbalanced Data. In: *2015 Seventh International Conference on Knowledge and Systems Engineering (KSE)*. 2015, pp. 280–285. Available from DOI: 10.1109/KSE.2015.24.

22. JO, Duke Taeho; JAPKOWICZ, Nathalie. Class imbalances versus small disjuncts. *SIGKDD Explorations*. 2004, vol. 6, pp. 40–49. Available from DOI: 10.1145/1007730.1007737.

23. CIESLAK, David; CHAWLA, Nitesh; STRIEGEL, Aaron. Combating imbalance in network intrusion datasets. In: 2006, pp. 732–737. Available from DOI: 10.1109/GRC.2006.1635905.

24. BARUA, Sukarna; ISLAM, Md. Monirul; YAO, Xin; MURASE, Kazuyuki. MWMOTE–Majority Weighted Minority Oversampling Technique for Imbalanced Data Set Learning. *IEEE Transactions on Knowledge and Data Engineering*. 2014, vol. 26, no. 2, pp. 405–425. Available from DOI: `10.1109/TKDE.2012.232`.

25. DOUZAS, Georgios; BACAO, Fernando; LAST, Felix. Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE. *Information Sciences*. 2018, vol. 465, pp. 1–20. ISSN 0020-0255. Available from DOI: `https://doi.org/10.1016/j.ins.2018.06.056`.

26. CHAWLA, Nitesh V.; LAZAREVIC, Aleksandar; HALL, Lawrence O.; BOWYER, Kevin W. SMOTEBoost: Improving Prediction of the Minority Class in Boosting. In: LAVRAČ, Nada; GAMBERGER, Dragan; TODOROVSKI, Ljupčo; BLOCKEEL, Hendrik (eds.). *Knowledge Discovery in Databases: PKDD 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 107–119. ISBN 978-3-540-39804-2.

27. FREUND, Yoav; SCHAPIRE, Robert E. Experiments with a New Boosting Algorithm. In: *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*. Bari, Italy: Morgan Kaufmann Publishers Inc., 1996, pp. 148–156. ICML'96. ISBN 1558604197.

28. GUO, Hongyu; VIKTOR, Herna. Learning from imbalanced data sets with boosting and data generation: The DataBoost-IM approach. *SIGKDD Explorations*. 2004, vol. 6, pp. 30–39. Available from DOI: `10.1145/1007730.1007736`.

29. CHEN, Sheng; HE, Haibo; GARCIA, Edwardo A. RAMOBoost: Ranked Minority Oversampling in Boosting. *IEEE Transactions on Neural Networks*. 2010, vol. 21, no. 10, pp. 1624–1642. Available from DOI: `10.1109/TNN.2010.2066988`.

30. PARK, Noseong; MOHAMMADI, Mahmoud; GORDE, Kshitij; JAJODIA, Sushil; PARK, Hongkyu; KIM, Youngmin. Data synthesis based on generative adversarial networks. *Proceedings of the VLDB Endowment*. 2018, vol. 11, no. 10, pp. 1071–1083. ISSN 2150-8097. Available from DOI: `10.14778/3231751.3231757`.

31. RADFORD, Alec; METZ, Luke; CHINTALA, Soumith. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. Available from arXiv: `1511.06434 [cs.LG]`.

32. KIM, Jayoung; LEE, Chaejeong; SHIN, Yehjin; PARK, Sewon; KIM, Minjung; PARK, Noseong; CHO, Jihoon. *SOS: Score-based Oversampling for Tabular Data*. 2022. Available from arXiv: `2206.08555 [cs.LG]`.

33. XU, Lei; VEERAMACHANENI, Kalyan. *Synthesizing Tabular Data using Generative Adversarial Networks*. 2018. Available from arXiv: `1811.11264 [cs.LG]`.

34. QUINTANA, Matias; MILLER, Clayton. Towards Class-Balancing Human Comfort Datasets with GANs. 2019. Available from DOI: `10.1145/3360322.3361016`.

35. XU, Lei; SKOULARIDOU, Maria; CUESTA-INFANTE, Alfredo; VEERAMACHANENI, Kalyan. *Modeling Tabular data using Conditional GAN*. 2019. Available from arXiv: `1907.00503 [cs.LG]`.

36. EOM, Gayeong; BYEON, Haewon. Searching for Optimal Oversampling to Process Imbalanced Data: Generative Adversarial Networks and Synthetic Minority Over-Sampling Technique. *Mathematics*. 2023, vol. 11, no. 16. ISSN 2227-7390. Available from DOI: `10.3390/math11163605`.

37. ENGELMANN, Justin; LESSMANN, Stefan. Conditional Wasserstein GAN-based oversampling of tabular data for imbalanced learning. *Expert Systems with Applications*. 2021, vol. 174, p. 114582. ISSN 0957-4174. Available from DOI: `https://doi.org/10.1016/j.eswa.2021.114582`.

38. MIRZA, Mehdi; OSINDERO, Simon. *Conditional Generative Adversarial Nets.* 2014. Available from arXiv: `1411.1784 [cs.LG]`.

39. ARJOVSKY, Martin; CHINTALA, Soumith; BOTTOU, Léon. *Wasserstein GAN.* 2017. Available from arXiv: `1701.07875 [stat.ML]`.

40. JANG, Eric; GU, Shixiang; POOLE, Ben. *Categorical Reparameterization with Gumbel-Softmax.* 2017. Available from arXiv: `1611.01144 [stat.ML]`.

41. GULRAJANI, Ishaan; AHMED, Faruk; ARJOVSKY, Martin; DUMOULIN, Vincent; COURVILLE, Aaron. *Improved Training of Wasserstein GANs.* 2017. Available from arXiv: `1704.00028 [cs.LG]`.

42. ZHAO, Zilong; KUNAR, Aditya; BIRKE, Robert; CHEN, Lydia Y. CTAB-GAN: Effective Table Data Synthesizing. In: BALASUBRAMANIAN, Vineeth N.; TSANG, Ivor (eds.). *Proceedings of The 13th Asian Conference on Machine Learning.* PMLR, 2021, vol. 157, pp. 97–112. Proceedings of Machine Learning Research. Available also from: `https://proceedings.mlr.press/v157/zhao21a.html`.

43. GRADSTEIN, Jonathan; SALHOV, Moshe; TULPAN, Yoav; LINDENBAUM, Ofir; AVERBUCH, Amir. *Imbalanced Classification via a Tabular Translation GAN.* 2022. Available from arXiv: `2204.08683 [cs.LG]`.

44. ZHU, Jun-Yan; PARK, Taesung; ISOLA, Phillip; EFROS, Alexei A. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks.* 2020. Available from arXiv: `1703.10593 [cs.CV]`.

45. DARABI, Sajad; ELOR, Yotam. *Synthesising Multi-Modal Minority Samples for Tabular Data.* 2021. Available from arXiv: `2105.08204 [cs.LG]`.

46. AI, Qingzhong; WANG, Pengyun; HE, Lirong; WEN, Liangjian; PAN, Lujia; XU, Zenglin. *Generative Oversampling for Imbalanced Data via Majority-Guided VAE.* 2023. Available from arXiv: `2302.10910 [cs.LG]`.

47. KIRKPATRICK, James; PASCANU, Razvan; RABINOWITZ, Neil; VENESS, Joel; DESJARDINS, Guillaume; RUSU, Andrei A.; MILAN, Kieran; QUAN, John; RAMALHO, Tiago; GRABSKA-BARWINSKA, Agnieszka; HASSABIS, Demis; CLOPATH, Claudia; KUMARAN, Dharshan; HADSELL, Raia. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences.* 2017, vol. 114, no. 13, pp. 3521–3526. ISSN 1091-6490. Available from DOI: `10.1073/pnas.1611835114`.

48. CHOI, Edward; BISWAL, Siddharth; MALIN, Bradley; DUKE, Jon; STEWART, Walter F.; SUN, Jimeng. *Generating Multi-label Discrete Patient Records using Generative Adversarial Networks.* 2018. Available from arXiv: `1703.06490 [cs.LG]`.

49. SHARMA, Anuraganand; SINGH, Prabhat Kumar; CHANDRA, Rohitash. SMOTified-GAN for Class Imbalanced Pattern Classification Problems. *IEEE Access.* 2022, vol. 10, pp. 30655–30665. ISSN 2169-3536. Available from DOI: `10.1109/access.2022.3158977`.

50. KOVÁCS, György. smote-variants: a Python Implementation of 85 Minority Oversampling Techniques. *Neurocomputing.* 2019, vol. 366, pp. 352–354. Available from DOI: `10.1016/j.neucom.2019.06.100`. (IF-2019=4.07).

51. FIGUEIRA, Alvaro; VAZ, Bruno. Survey on Synthetic Data Generation, Evaluation Methods and GANs. *Mathematics.* 2022, vol. 10, no. 15. ISSN 2227-7390. Available from DOI: `10.3390/math10152733`.

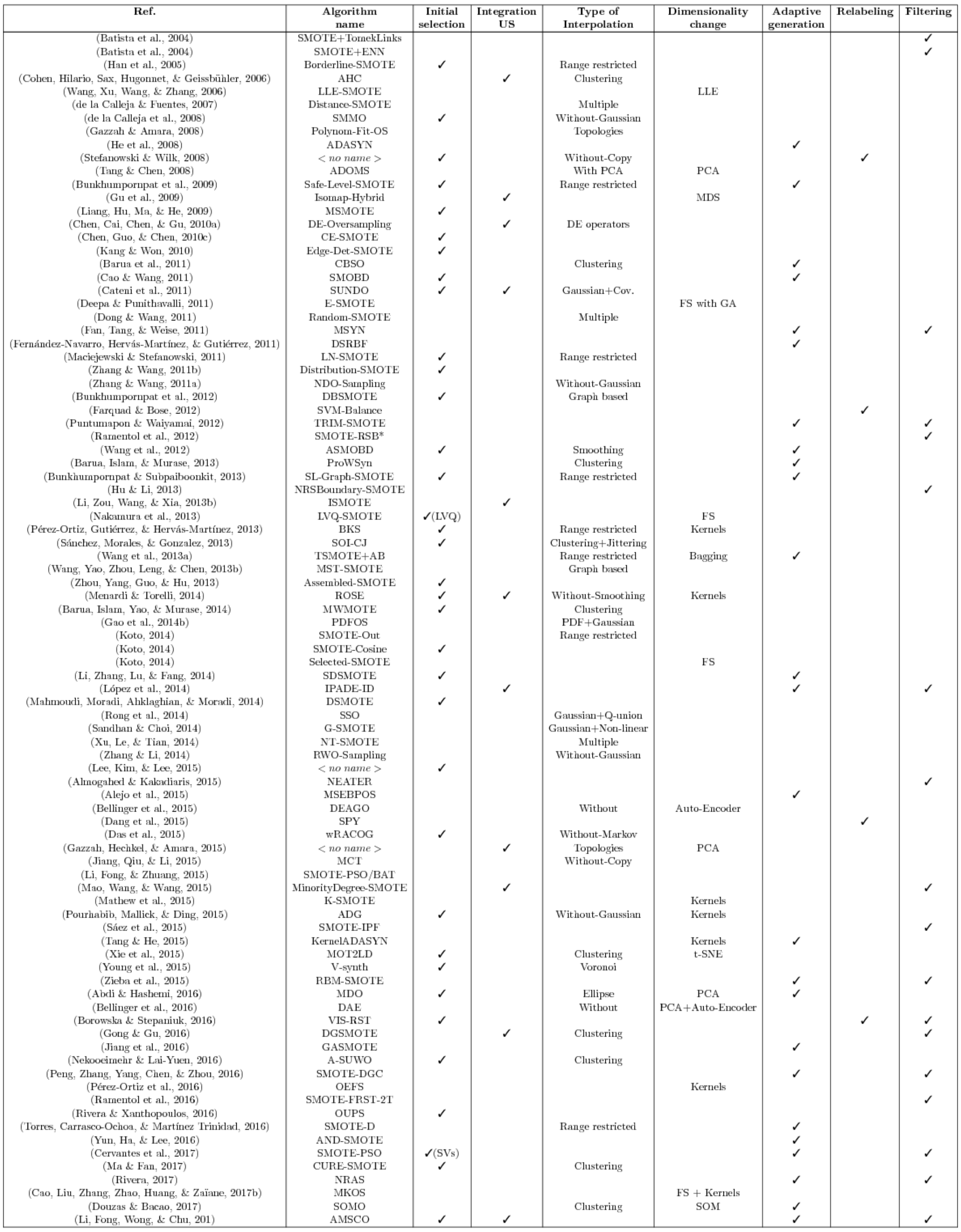
52. LIN, Zinan; KHETAN, Ashish; FANTI, Giulia; OH, Sewoong. *PacGAN: The power of two samples in generative adversarial networks.* 2018. Available from arXiv: `1712.04086 [cs.LG]`.

53. ALCALA-FDEZ, Jesus; FERNÁNDEZ, Alberto; LUENGO, Julián; DERRAC, J.; GARC'IA, S; SANCHEZ, Luciano; HERRERA, Francisco. KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. *Journal of Multiple-Valued Logic and Soft Computing*. 2010, vol. 17, pp. 255–287.

54. HOFMANN, Hans. *UCI Machine Learning Repository* [UCI Machine Learning Repository]. 1994. DOI: https://doi.org/10.24432/C5NC77.

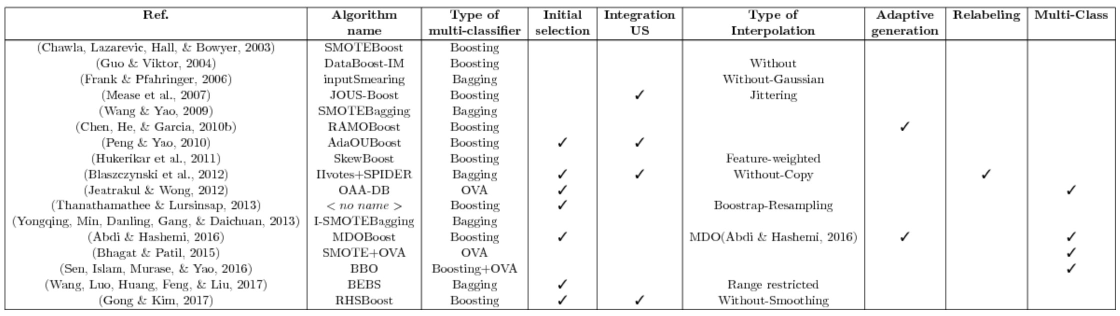
55. GALA, Nirav Bharat. *Generation of synthetic examples for imbalanced tabular data*. 2023. PhD thesis. Dublin, National College of Ireland.

Appendix A
Figures



Ref.	Algorithm name	Initial selection	Integration US	Type of Interpolation	Dimensionality change	Adaptive generation	Relabeling	Filtering
(Batista et al., 2004)	SMOTE+TomekLinks							✓
(Batista et al., 2004)	SMOTE+ENN							✓
(Han et al., 2005)	Borderline-SMOTE	✓		Range restricted				
(Cohen, Hilario, Sax, Hugonnet, & Geissbühler, 2006)	AHC		✓	Clustering				
(Wang, Xu, Wang, & Zhang, 2006)	LLE-SMOTE				LLE			
(de la Calleja & Fuentes, 2007)	Distance-SMOTE			Multiple				
(de la Calleja et al., 2008)	SMMO	✓		Without-Gaussian				
(Gazzah & Amara, 2008)	Polynom-Fit-OS			Topologies				
(He et al., 2008)	ADASYN					✓		
(Stefanowski & Wilk, 2008)	< *no name* >	✓		Without-Copy			✓	
(Tang & Chen, 2008)	ADOMS			With PCA	PCA			
(Bunkhumpornpat et al., 2009)	Safe-Level-SMOTE	✓		Range restricted		✓		
(Gu et al., 2009)	Isomap-Hybrid		✓		MDS			
(Liang, Hu, Ma, & He, 2009)	MSMOTE	✓						
(Chen, Cai, Chen, & Gu, 2010a)	DE-Oversampling		✓	DE operators				
(Chen, Guo, & Chen, 2010c)	CE-SMOTE	✓						
(Kang & Won, 2010)	Edge-Det-SMOTE	✓						
(Barua et al., 2011)	CBSO			Clustering		✓		
(Cao & Wang, 2011)	SMOBD	✓				✓		
(Cateni et al., 2011)	SUNDO	✓	✓	Gaussian+Cov.				
(Deepa & Punithavalli, 2011)	E-SMOTE				FS with GA			
(Dong & Wang, 2011)	Random-SMOTE			Multiple				
(Fan, Tang, & Weise, 2011)	MSYN					✓		✓
(Fernández-Navarro, Hervás-Martínez, & Gutiérrez, 2011)	DSRBF					✓		
(Maciejewski & Stefanowski, 2011)	LN-SMOTE	✓		Range restricted				
(Zhang & Wang, 2011b)	Distribution-SMOTE	✓						
(Zhang & Wang, 2011a)	NDO-Sampling			Without-Gaussian				
(Bunkhumpornpat et al., 2012)	DBSMOTE	✓		Graph based				
(Farquad & Bose, 2012)	SVM-Balance						✓	
(Puntumapon & Waiyamai, 2012)	TRIM-SMOTE					✓		✓
(Ramentol et al., 2012)	SMOTE-RSB*							✓
(Wang et al., 2012)	ASMOBD	✓		Smoothing		✓		
(Barua, Islam, & Murase, 2013)	ProWSyn			Clustering		✓		
(Bunkhumpornpat & Subpaiboonkit, 2013)	SL-Graph-SMOTE	✓		Range restricted		✓		
(Hu & Li, 2013)	NRSBoundary-SMOTE							✓
(Li, Zou, Wang, & Xia, 2013b)	ISMOTE		✓					
(Nakamura et al., 2013)	LVQ-SMOTE	✓(LVQ)			FS			
(Pérez-Ortiz, Gutiérrez, & Hervás-Martínez, 2013)	BKS	✓		Range restricted	Kernels			
(Sánchez, Morales, & Gonzalez, 2013)	SOI-CJ	✓		Clustering+Jittering				
(Wang et al., 2013a)	TSMOTE+AB			Range restricted	Bagging	✓		
(Wang, Yao, Zhou, Leng, & Chen, 2013b)	MST-SMOTE			Graph based				
(Zhou, Yang, Guo, & Hu, 2013)	Assembled-SMOTE	✓						
(Menardi & Torelli, 2014)	ROSE	✓	✓	Without-Smoothing	Kernels			
(Barua, Islam, Yao, & Murase, 2014)	MWMOTE	✓		Clustering				
(Gao et al., 2014b)	PDFOS			PDF+Gaussian				
(Koto, 2014)	SMOTE-Out			Range restricted				
(Koto, 2014)	SMOTE-Cosine	✓						
(Koto, 2014)	Selected-SMOTE				FS			
(Li, Zhang, Lu, & Fang, 2014)	SDSMOTE	✓				✓		
(López et al., 2014)	IPADE-ID		✓			✓		✓
(Mahmoudi, Moradi, Ahklaghian, & Moradi, 2014)	DSMOTE	✓						
(Rong et al., 2014)	SSO			Gaussian+Q-union				
(Sandhan & Choi, 2014)	G-SMOTE			Gaussian+Non-linear				
(Xu, Le, & Tian, 2014)	NT-SMOTE			Multiple				
(Zhang & Li, 2014)	RWO-Sampling			Without-Gaussian				
(Lee, Kim, & Lee, 2015)	< *no name* >	✓						
(Almogahed & Kakadiaris, 2015)	NEATER							✓
(Alejo et al., 2015)	MSEBPOS					✓		
(Bellinger et al., 2015)	DEAGO			Without	Auto-Encoder			
(Dang et al., 2015)	SPY						✓	
(Das et al., 2015)	wRACOG	✓		Without-Markov				
(Gazzah, Hechkel, & Amara, 2015)	< *no name* >		✓	Topologies	PCA			
(Jiang, Qiu, & Li, 2015)	MCT			Without-Copy				
(Li, Fong, & Zhuang, 2015)	SMOTE-PSO/BAT							
(Mao, Wang, & Wang, 2015)	MinorityDegree-SMOTE		✓					✓
(Mathew et al., 2015)	K-SMOTE				Kernels			
(Pourhabib, Mallick, & Ding, 2015)	ADG	✓		Without-Gaussian	Kernels			
(Sáez et al., 2015)	SMOTE-IPF							✓
(Tang & He, 2015)	KernelADASYN				Kernels	✓		
(Xie et al., 2015)	MOT2LD	✓		Clustering	t-SNE			
(Young et al., 2015)	V-synth	✓		Voronoi				
(Zieba et al., 2015)	RBM-SMOTE					✓		✓
(Abdi & Hashemi, 2016)	MDO	✓		Ellipse	PCA	✓		
(Bellinger et al., 2016)	DAE			Without	PCA+Auto-Encoder			
(Borowska & Stepaniuk, 2016)	VIS-RST	✓					✓	✓
(Gong & Gu, 2016)	DGSMOTE		✓	Clustering				✓
(Jiang et al., 2016)	GASMOTE					✓		
(Nekooeimehr & Lai-Yuen, 2016)	A-SUWO	✓		Clustering				
(Peng, Zhang, Yang, Chen, & Zhou, 2016)	SMOTE-DGC					✓		✓
(Pérez-Ortiz et al., 2016)	OEFS				Kernels			
(Ramentol et al., 2016)	SMOTE-FRST-2T							✓
(Rivera & Xanthopoulos, 2016)	OUPS	✓						
(Torres, Carrasco-Ochoa, & Martínez Trinidad, 2016)	SMOTE-D			Range restricted		✓		
(Yun, Ha, & Lee, 2016)	AND-SMOTE					✓		
(Cervantes et al., 2017)	SMOTE-PSO	✓(SVs)				✓		✓
(Ma & Fan, 2017)	CURE-SMOTE	✓		Clustering				
(Rivera, 2017)	NRAS					✓		✓
(Cao, Liu, Zhang, Zhao, Huang, & Zaïane, 2017b)	MKOS				FS + Kernels			
(Douzas & Bacao, 2017)	SOMO			Clustering	SOM	✓		
(Li, Fong, Wong, & Chu, 201)	AMSCO	✓	✓			✓		✓

Figure A.1 SMOTE algorithm extensions taken from [16]



Ref.	Algorithm name	Type of multi-classifier	Initial selection	Integration US	Type of Interpolation	Adaptive generation	Relabeling	Multi-Class
(Chawla, Lazarevic, Hall, & Bowyer, 2003)	SMOTEBoost	Boosting						
(Guo & Viktor, 2004)	DataBoost-IM	Boosting			Without			
(Frank & Pfahringer, 2006)	inputSmearing	Bagging			Without-Gaussian			
(Mease et al., 2007)	JOUS-Boost	Boosting		✓	Jittering			
(Wang & Yao, 2009)	SMOTEBagging	Bagging						
(Chen, He, & Garcia, 2010b)	RAMOBoost	Boosting				✓		
(Peng & Yao, 2010)	AdaOUBoost	Boosting	✓	✓				
(Hukerikar et al., 2011)	SkewBoost	Boosting			Feature-weighted			
(Blaszczynski et al., 2012)	IIvotes+SPIDER	Bagging	✓	✓	Without-Copy		✓	
(Jeatrakul & Wong, 2012)	OAA-DB	OVA	✓					✓
(Thanathamathee & Lursinsap, 2013)	< *no name* >	Boosting	✓		Boostrap-Resampling			
(Yongqing, Min, Danling, Gang, & Daichuan, 2013)	I-SMOTEBagging	Bagging						
(Abdi & Hashemi, 2016)	MDOBoost	Boosting	✓		MDO(Abdi & Hashemi, 2016)	✓		✓
(Bhagat & Patil, 2015)	SMOTE+OVA	OVA						✓
(Sen, Islam, Murase, & Yao, 2016)	BBO	Boosting+OVA						✓
(Wang, Luo, Huang, Feng, & Liu, 2017)	BEBS	Bagging	✓		Range restricted			
(Gong & Kim, 2017)	RHSBoost	Boosting	✓	✓	Without-Smoothing			

Figure A.2 SMOTE-based ensemble methods taken from [16]

Appendix B

Tables

dataset	none	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
pima	MLP 0.738 ±.021	MLP 0.733 ±.034 (-0.005)	SVM 0.758 ±.026 (0.02)	LOGIT 0.755 ±.043 (0.017)	**MLP** **0.761** **±.03** **(0.023)**	MLP 0.74 ±.026 (0.002)	SVM 0.75 ±.036 (0.012)	LOGIT 0.737 ±.037 (-0.001)	LOGIT 0.75 ±.027 (0.012)	SVM 0.747 ±.049 (0.009)	RFC 0.758 ±.034 (0.02)	MLP 0.754 ±.033 (0.016)
german	SVM 0.662 ±.014	LOGIT 0.691 ±.021 (0.029)	SVM 0.706 ±.022 (0.044)	**SVM** **0.723** **±.017** **(0.061)**	LOGIT 0.699 ±.028 (0.037)	LOGIT 0.644 ±.0 (-0.018)	LOGIT 0.712 ±.031 (0.05)	RFC 0.688 ±.03 (0.026)	LOGIT 0.696 ±.012 (0.034)	SVM 0.709 ±.022 (0.047)	LOGIT 0.68 ±.018 (0.018)	LOGIT 0.674 ±.016 (0.012)
haberman	RFC 0.59 ±.037	RFC 0.624 ±.069 (0.034)	MLP 0.66 ±.024 (0.07)	MLP 0.653 ±.051 (0.063)	**MLP** **0.664** **±.05** **(0.074)**	LOGIT 0.596 ±.022 (0.006)	MLP 0.662 ±.023 (0.072)	MLP 0.625 ±.008 (0.035)	RFC 0.58 ±.0 (-0.01)	MLP 0.637 ±.006 (0.047)	RFC 0.652 ±.055 (0.062)	MLP 0.635 ±.046 (0.045)
adult	RFC 0.777 ±.001	LOGIT 0.814 ±.002 (0.037)	**LOGIT** **0.822** **±.001** **(0.045)**	LOGIT 0.816 ±.001 (0.039)	SVM 0.82 ±.002 (0.043)	RFC 0.781 ±.001 (0.004)	**LOGIT** **0.822** **±.001** **(0.045)**	LOGIT 0.809 ±.007 (0.032)	LOGIT 0.802 ±.003 (0.025)	LOGIT 0.819 ±.001 (0.042)	LOGIT 0.81 ±.002 (0.033)	LOGIT 0.788 ±.004 (0.011)
yeast3	MLP 0.867 ±.032	MLP 0.899 ±.02 (0.032)	KNN 0.901 ±.013 (0.034)	LOGIT 0.888 ±.011 (0.021)	**KNN** **0.908** **±.03** **(0.041)**	KNN 0.889 ±.019 (0.022)	MLP 0.895 ±.009 (0.028)	RFC 0.881 ±.036 (0.014)	SVM 0.885 ±.009 (0.018)	LOGIT 0.888 ±.004 (0.021)	MLP 0.885 ±.003 (0.018)	MLP 0.858 ±.046 (-0.009)
abalone9-18	SVM 0.714 ±.046	SVM 0.822 ±.066 (0.108)	**SVM** **0.884** **±.035** **(0.17)**	SVM 0.852 ±.045 (0.138)	LOGIT 0.871 ±.06 (0.157)	nan	SVM 0.88 ±.034 (0.166)	SVM 0.781 ±.112 (0.067)	SVM 0.78 ±.031 (0.066)	SVM 0.775 ±.029 (0.061)	MLP 0.832 ±.066 (0.118)	LOGIT 0.836 ±.016 (0.122)
winequality-red-4	DTC 0.53 ±.018	LOGIT 0.733 ±.058 (0.203)	LOGIT 0.728 ±.041 (0.198)	**SVM** **0.761** **±.04** **(0.231)**	LOGIT 0.733 ±.037 (0.203)	LOGIT 0.691 ±.038 (0.161)	LOGIT 0.727 ±.043 (0.197)	LOGIT 0.682 ±.023 (0.152)	SVM 0.679 ±.021 (0.149)	LOGIT 0.661 ±.015 (0.131)	RFC 0.673 ±.049 (0.143)	SVM 0.622 ±.003 (0.092)
mammography	DTC 0.801 ±.009	MLP 0.901 ±.021 (0.1)	**MLP** **0.916** **±.007** **(0.115)**	MLP 0.903 ±.009 (0.102)	KNN 0.904 ±.011 (0.103)	SVM 0.863 ±.021 (0.062)	MLP 0.902 ±.009 (0.101)	MLP 0.893 ±.015 (0.092)	MLP 0.895 ±.016 (0.094)	LOGIT 0.871 ±.01 (0.07)	KNN 0.848 ±.019 (0.047)	LOGIT 0.83 ±.01 (0.029)
abalone-20-vs-8-9-10	MLP 0.645 ±.03	**LOGIT** **0.894** **±.06** **(0.249)**	LOGIT 0.851 ±.05 (0.206)	SVM 0.858 ±.053 (0.213)	LOGIT 0.851 ±.049 (0.206)	LOGIT 0.803 ±.0 (0.158)	LOGIT 0.865 ±.028 (0.22)	MLP 0.74 ±.004 (0.095)	DTC 0.697 ±.03 (0.052)	LOGIT 0.681 ±.103 (0.036)	SVM 0.823 ±.03 (0.178)	LOGIT 0.813 ±.033 (0.168)

Table B.1 Balanced Accuracy without Classifiers HPs Tuning: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

dataset	none	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
pima	RFC 0.722 ±.023	MLP 0.724 ±.013 (0.002)	LOGIT 0.754 ±.037 (0.032)	**LOGIT** **0.756** **±.044** **(0.034)**	RFC 0.753 ±.029 (0.031)	RFC 0.743 ±.024 (0.021)	SVM 0.751 ±.035 (0.029)	SVM 0.738 ±.037 (0.016)	LOGIT 0.75 ±.027 (0.028)	LOGIT 0.744 ±.05 (0.022)	MLP 0.742 ±.016 (0.02)	LOGIT 0.749 ±.041 (0.027)
german	MLP 0.688 ±.014	SVM 0.692 ±.02 (0.004)	SVM 0.704 ±.019 (0.016)	**SVM** **0.724** **±.015** **(0.036)**	LOGIT 0.698 ±.027 (0.01)	LOGIT 0.642 ±.0 (-0.046)	LOGIT 0.713 ±.031 (0.025)	RFC 0.696 ±.027 (0.008)	LOGIT 0.696 ±.012 (0.008)	SVM 0.707 ±.015 (0.019)	MLP 0.678 ±.025 (-0.01)	LOGIT 0.672 ±.014 (-0.016)
haberman	RFC 0.593 ±.032	MLP 0.627 ±.026 (0.034)	MLP 0.627 ±.06 (0.034)	MLP 0.633 ±.064 (0.04)	MLP 0.655 ±.063 (0.062)	LOGIT 0.606 ±.02 (0.013)	MLP 0.618 ±.052 (0.025)	RFC 0.631 ±.008 (0.038)	DTC 0.58 ±.0 (-0.013)	RFC 0.648 ±.037 (0.055)	**RFC** **0.667** **±.036** **(0.074)**	MLP 0.64 ±.04 (0.047)
adult	RFC 0.778 ±.001	LOGIT 0.814 ±.002 (0.036)	LOGIT 0.821 ±.001 (0.043)	LOGIT 0.816 ±.001 (0.038)	SVM 0.821 ±.002 (0.043)	RFC 0.781 ±.001 (0.003)	**LOGIT** **0.822** **±.001** **(0.044)**	SVM 0.81 ±.006 (0.032)	LOGIT 0.802 ±.003 (0.024)	LOGIT 0.819 ±.001 (0.041)	SVM 0.81 ±.002 (0.032)	LOGIT 0.782 ±.008 (0.004)
yeast3	DTC 0.89 ±.036	MLP 0.906 ±.013 (0.016)	KNN 0.894 ±.025 (0.004)	LOGIT 0.887 ±.01 (-0.003)	**DTC** **0.909** **±.022** **(0.019)**	SVM 0.898 ±.016 (0.008)	SVM 0.896 ±.021 (0.006)	DTC 0.894 ±.031 (0.004)	LOGIT 0.887 ±.007 (-0.003)	DTC 0.905 ±.036 (0.015)	DTC 0.905 ±.03 (0.015)	DTC 0.893 ±.031 (0.003)
abalone9-18	LOGIT 0.764 ±.061	LOGIT 0.818 ±.063 (0.054)	LOGIT 0.882 ±.035 (0.118)	SVM 0.868 ±.053 (0.104)	LOGIT 0.859 ±.022 (0.095)	nan	**SVM** **0.889** **±.038** **(0.125)**	LOGIT 0.775 ±.112 (0.011)	SVM 0.78 ±.031 (0.016)	LOGIT 0.763 ±.038 (-0.001)	MLP 0.825 ±.059 (0.061)	LOGIT 0.751 ±.064 (-0.013)
winequality-red-4	KNN 0.536 ±.015	LOGIT 0.733 ±.058 (0.197)	SVM 0.749 ±.015 (0.213)	**LOGIT** **0.754** **±.038** **(0.218)**	SVM 0.737 ±.019 (0.201)	LOGIT 0.692 ±.039 (0.156)	SVM 0.741 ±.033 (0.205)	LOGIT 0.682 ±.023 (0.146)	SVM 0.678 ±.032 (0.142)	LOGIT 0.661 ±.014 (0.125)	MLP 0.673 ±.055 (0.137)	LOGIT 0.615 ±.015 (0.079)
mammography	KNN 0.791 ±.019	MLP 0.899 ±.011 (0.108)	**MLP** **0.911** **±.008** **(0.12)**	MLP 0.898 ±.007 (0.107)	KNN 0.906 ±.012 (0.115)	SVM 0.873 ±.02 (0.082)	MLP 0.907 ±.006 (0.116)	MLP 0.895 ±.02 (0.104)	MLP 0.895 ±.017 (0.104)	LOGIT 0.871 ±.01 (0.08)	LOGIT 0.847 ±.024 (0.056)	LOGIT 0.83 ±.01 (0.039)
abalone-20-vs-8-9-10	LOGIT 0.645 ±.029	**SVM** **0.901** **±.059** **(0.256)**	SVM 0.85 ±.0 (0.205)	SVM 0.857 ±.054 (0.212)	SVM 0.867 ±.027 (0.222)	LOGIT 0.74 ±.0 (0.095)	LOGIT 0.866 ±.028 (0.221)	MLP 0.739 ±.005 (0.094)	MLP 0.659 ±.024 (0.014)	LOGIT 0.683 ±.103 (0.038)	LOGIT 0.822 ±.031 (0.177)	MLP 0.686 ±.05 (0.041)

Table B.2 Balanced Accuracy with Classifiers HPs Tuning: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

results	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
better	8	9	9	9	7	9	8	8	9	9	8
same	0	0	0	0	0	0	0	0	0	0	0
worse	1	0	0	0	1	0	1	1	0	0	1
avg diff [%]	8.74	10.02	9.83	9.86	4.96	9.9	5.69	4.89	5.16	7.08	5.4
avg std	0.016	0.001	0.007	0.01	-0.004	0.001	0.007	-0.007	0.003	0.008	-0.0

Table B.3 Balanced Accuracy without Classifiers HPs Tuning Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse result than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

results	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
better	9	9	8	9	7	9	9	7	8	8	7
same	0	0	0	0	0	0	0	0	0	0	0
worse	0	0	1	0	1	0	0	2	1	1	2
avg diff [%]	7.86	8.72	8.73	8.87	4.15	8.84	5.03	3.56	4.38	6.24	2.34
avg std	0.004	-0.003	0.006	-0.001	-0.006	0.002	0.004	-0.009	0.008	0.005	0.005

Table B.4 Balanced Accuracy with Classifiers HPs Tuning Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse result than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

dataset	none	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
pima	**MLP 0.71 ±.077**	MLP 0.641 ±.079 (-0.069)	RFC 0.66 ±.058 (-0.05)	RFC 0.634 ±.074 (-0.076)	RFC 0.666 ±.054 (-0.044)	RFC 0.672 ±.068 (-0.038)	RFC 0.671 ±.071 (-0.039)	RFC 0.625 ±.043 (-0.085)	LOGIT 0.656 ±.063 (-0.054)	RFC 0.647 ±.055 (-0.063)	RFC 0.647 ±.07 (-0.063)	RFC 0.7 ±.065 (-0.01)
german	RFC 0.673 ±.02	RFC 0.575 ±.01 (-0.098)	RFC 0.613 ±.016 (-0.06)	RFC 0.626 ±.033 (-0.047)	RFC 0.651 ±.032 (-0.022)	RFC 0.66 ±.0 (-0.013)	RFC 0.616 ±.009 (-0.057)	RFC 0.519 ±.027 (-0.154)	RFC 0.595 ±.048 (-0.078)	RFC 0.634 ±.043 (-0.039)	RFC 0.555 ±.023 (-0.118)	**RFC 0.679 ±.032 (0.006)**
haberman	**SVM 0.72 ±.227**	MLP 0.494 ±.046 (-0.226)	MLP 0.484 ±.082 (-0.236)	MLP 0.485 ±.119 (-0.235)	MLP 0.522 ±.097 (-0.198)	SVM 0.513 ±.11 (-0.207)	MLP 0.5 ±.071 (-0.22)	MLP 0.458 ±.051 (-0.262)	MLP 0.353 ±.0 (-0.367)	LOGIT 0.435 ±.095 (-0.285)	RFC 0.447 ±.094 (-0.273)	MLP 0.513 ±.126 (-0.207)
adult	**LOGIT 0.74 ±.011**	RFC 0.704 ±.005 (-0.036)	RFC 0.659 ±.002 (-0.081)	RFC 0.665 ±.006 (-0.075)	RFC 0.712 ±.003 (-0.028)	RFC 0.718 ±.006 (-0.022)	RFC 0.667 ±.002 (-0.073)	RFC 0.684 ±.005 (-0.056)	RFC 0.663 ±.013 (-0.077)	RFC 0.721 ±.001 (-0.019)	RFC 0.619 ±.01 (-0.121)	RFC 0.729 ±.001 (-0.011)
yeast3	**KNN 0.827 ±.018**	RFC 0.79 ±.046 (-0.037)	RFC 0.747 ±.065 (-0.08)	DTC 0.674 ±.046 (-0.153)	MLP 0.714 ±.035 (-0.113)	RFC 0.738 ±.042 (-0.089)	RFC 0.764 ±.069 (-0.063)	RFC 0.794 ±.06 (-0.033)	RFC 0.797 ±.063 (-0.03)	RFC 0.816 ±.059 (-0.011)	RFC 0.772 ±.086 (-0.055)	RFC 0.796 ±.029 (-0.031)
abalone9-18	**LOGIT 1.0 ±.0**	MLP 0.535 ±.161 (-0.465)	SVM 0.411 ±.048 (-0.589)	RFC 0.46 ±.134 (-0.54)	MLP 0.438 ±.051 (-0.562)	nan	RFC 0.556 ±.08 (-0.444)	MLP 0.646 ±.172 (-0.354)	SVM 0.415 ±.059 (-0.585)	RFC 0.218 ±.043 (-0.782)	MLP 0.38 ±.059 (-0.62)	MLP 0.768 ±.076 (-0.232)
winequality-red-4	**KNN 0.333 ±.471**	RFC 0.187 ±.035 (-0.146)	RFC 0.139 ±.12 (-0.194)	MLP 0.147 ±.047 (-0.186)	MLP 0.111 ±.067 (-0.222)	RFC 0.167 ±.236 (-0.166)	MLP 0.156 ±.024 (-0.177)	KNN 0.263 ±.13 (-0.07)	RFC 0.166 ±.007 (-0.167)	DTC 0.111 ±.016 (-0.222)	KNN 0.17 ±.06 (-0.163)	KNN 0.143 ±.202 (-0.19)
mammography	**SVM 0.875 ±.102**	RFC 0.558 ±.076 (-0.317)	RFC 0.585 ±.012 (-0.29)	RFC 0.392 ±.027 (-0.483)	RFC 0.476 ±.036 (-0.399)	RFC 0.737 ±.037 (-0.138)	RFC 0.793 ±.026 (-0.082)	RFC 0.297 ±.051 (-0.578)	RFC 0.537 ±.122 (-0.338)	RFC 0.74 ±.013 (-0.135)	RFC 0.388 ±.055 (-0.487)	RFC 0.842 ±.027 (-0.033)
abalone-20-vs-8-9-10	SVM 0.778 ±.157	MLP 0.505 ±.217 (-0.273)	RFC 0.296 ±.078 (-0.482)	MLP 0.356 ±.05 (-0.422)	MLP 0.29 ±.034 (-0.488)	RFC 0.667 ±.0 (-0.111)	RFC 0.556 ±.079 (-0.222)	MLP 0.28 ±.071 (-0.498)	SVM 0.438 ±.401 (-0.34)	DTC 0.082 ±.019 (-0.696)	MLP 0.156 ±.053 (-0.622)	**MLP 0.783 ±.165 (0.005)**

Table B.5 Precision without Classifiers HPs Tuning: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

dataset	none	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
pima	**SVM 0.717 ±.098**	RFC 0.647 ±.102 (-0.07)	RFC 0.671 ±.068 (-0.046)	RFC 0.638 ±.072 (-0.079)	MLP 0.653 ±.06 (-0.064)	RFC 0.678 ±.072 (-0.039)	RFC 0.664 ±.061 (-0.053)	RFC 0.626 ±.039 (-0.091)	LOGIT 0.656 ±.063 (-0.061)	RFC 0.625 ±.04 (-0.092)	SVM 0.63 ±.064 (-0.087)	RFC 0.69 ±.086 (-0.027)
german	RFC 0.658 ±.026	RFC 0.545 ±.042 (-0.113)	RFC 0.599 ±.028 (-0.059)	RFC 0.631 ±.021 (-0.027)	RFC 0.646 ±.042 (-0.012)	RFC 0.63 ±.0 (-0.028)	RFC 0.603 ±.019 (-0.055)	RFC 0.525 ±.027 (-0.133)	MLP 0.551 ±.054 (-0.107)	RFC 0.643 ±.014 (-0.015)	RFC 0.557 ±.021 (-0.101)	**RFC 0.694 ±.044 (0.036)**
haberman	**SVM 0.72 ±.227**	MLP 0.514 ±.066 (-0.206)	MLP 0.452 ±.12 (-0.268)	MLP 0.461 ±.136 (-0.259)	MLP 0.508 ±.112 (-0.212)	RFC 0.555 ±.124 (-0.165)	MLP 0.435 ±.084 (-0.285)	RFC 0.444 ±.063 (-0.276)	DTC 0.351 ±.0 (-0.369)	RFC 0.435 ±.053 (-0.285)	RFC 0.459 ±.068 (-0.261)	MLP 0.526 ±.115 (-0.194)
adult	LOGIT 0.741 ±.012	DTC 0.754 ±.006 (0.013)	RFC 0.687 ±.035 (-0.054)	RFC 0.695 ±.032 (-0.046)	RFC 0.722 ±.011 (-0.019)	RFC 0.727 ±.012 (-0.014)	RFC 0.667 ±.003 (-0.074)	DTC 0.734 ±.017 (-0.007)	RFC 0.666 ±.018 (-0.075)	DTC 0.765 ±.015 (0.024)	RFC 0.62 ±.012 (-0.121)	**DTC 0.796 ±.003 (0.055)**
yeast3	**MLP 0.817 ±.017**	RFC 0.774 ±.041 (-0.043)	RFC 0.716 ±.056 (-0.101)	RFC 0.692 ±.135 (-0.125)	RFC 0.742 ±.108 (-0.075)	RFC 0.733 ±.035 (-0.084)	RFC 0.76 ±.064 (-0.057)	RFC 0.797 ±.057 (-0.02)	RFC 0.787 ±.054 (-0.03)	RFC 0.799 ±.055 (-0.018)	RFC 0.786 ±.096 (-0.031)	RFC 0.807 ±.043 (-0.01)
abalone9-18	**MLP 0.87 ±.094**	MLP 0.499 ±.163 (-0.371)	MLP 0.449 ±.062 (-0.421)	RFC 0.429 ±.111 (-0.441)	MLP 0.472 ±.058 (-0.398)	nan	RFC 0.533 ±.065 (-0.337)	MLP 0.597 ±.197 (-0.273)	MLP 0.418 ±.091 (-0.452)	RFC 0.196 ±.072 (-0.674)	MLP 0.407 ±.044 (-0.463)	LOGIT 0.75 ±.068 (-0.12)
winequality-red-4	**KNN 0.236 ±.086**	RFC 0.201 ±.079 (-0.035)	MLP 0.162 ±.054 (-0.074)	RFC 0.172 ±.043 (-0.064)	LOGIT 0.101 ±.006 (-0.135)	RFC 0.233 ±.205 (-0.003)	MLP 0.189 ±.057 (-0.047)	KNN 0.187 ±.064 (-0.049)	RFC 0.178 ±.029 (-0.058)	DTC 0.099 ±.02 (-0.137)	KNN 0.175 ±.059 (-0.061)	KNN 0.149 ±.057 (-0.087)
mammography	**RFC 0.86 ±.012**	RFC 0.575 ±.075 (-0.285)	RFC 0.58 ±.005 (-0.28)	RFC 0.388 ±.019 (-0.472)	RFC 0.494 ±.034 (-0.366)	RFC 0.738 ±.037 (-0.122)	RFC 0.8 ±.03 (-0.06)	RFC 0.304 ±.045 (-0.556)	RFC 0.538 ±.121 (-0.322)	RFC 0.751 ±.042 (-0.109)	RFC 0.391 ±.061 (-0.469)	RFC 0.842 ±.027 (-0.018)
abalone-20-vs-8-9-10	**KNN 0.833 ±.236**	MLP 0.45 ±.156 (-0.383)	MLP 0.317 ±.061 (-0.516)	RFC 0.332 ±.026 (-0.501)	MLP 0.318 ±.076 (-0.515)	RFC 0.5 ±.0 (-0.333)	RFC 0.611 ±.079 (-0.222)	MLP 0.279 ±.083 (-0.554)	SVM 0.433 ±.403 (-0.4)	MLP 0.111 ±.041 (-0.722)	MLP 0.157 ±.052 (-0.676)	SVM 0.756 ±.175 (-0.077)

Table B.6 Precision with Classifiers HPs Tuning: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

results	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
better	0	0	0	0	0	0	0	0	0	0	2
same	0	0	0	0	0	0	0	0	0	0	0
worse	9	9	9	9	8	9	9	9	9	9	7
avg diff [%]	-18.52	-22.91	-24.63	-23.07	-9.8	-15.3	-23.22	-22.62	-25.02	-28.02	-7.81
avg std	-0.045	-0.067	-0.061	-0.075	-0.073	-0.072	-0.053	-0.034	-0.082	-0.064	-0.04

Table B.7 Precision without Classifiers HPs Tuning Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse result than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

results	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
better	1	0	0	0	0	0	0	0	1	0	2
same	0	0	0	0	0	0	0	0	0	0	0
worse	8	9	9	9	8	9	9	9	8	9	7
avg diff [%]	-16.59	-20.21	-22.38	-19.96	-9.85	-13.22	-21.77	-20.82	-22.53	-25.22	-4.91
avg std	-0.009	-0.035	-0.024	-0.033	-0.029	-0.038	-0.024	0.003	-0.051	-0.037	-0.021

Table B.8 Precision with Classifiers HPs Tuning Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse result than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

dataset	none	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
pima	DTC 0.63 ±.03	MLP 0.679 ±.03 (0.049)	LOGIT 0.733 ±.038 (0.103)	**LOGIT** **0.77** **±.029** **(0.14)**	MLP 0.728 ±.027 (0.098)	MLP 0.675 ±.012 (0.045)	LOGIT 0.728 ±.027 (0.098)	LOGIT 0.745 ±.015 (0.115)	LOGIT 0.704 ±.0 (0.074)	LOGIT 0.749 ±.056 (0.119)	RFC 0.741 ±.017 (0.111)	SVM 0.728 ±.02 (0.098)
german	DTC 0.507 ±.01	SVM 0.615 ±.014 (0.108)	SVM 0.696 ±.021 (0.189)	**SVM** **0.759** **±.014** **(0.252)**	KNN 0.726 ±.037 (0.219)	LOGIT 0.489 ±.0 (-0.018)	SVM 0.711 ±.018 (0.204)	SVM 0.652 ±.05 (0.145)	LOGIT 0.615 ±.014 (0.108)	SVM 0.696 ±.019 (0.189)	LOGIT 0.604 ±.055 (0.097)	LOGIT 0.537 ±.026 (0.03)
haberman	DTC 0.403 ±.052	LOGIT 0.444 ±.071 (0.041)	MLP 0.569 ±.071 (0.166)	SVM 0.556 ±.02 (0.153)	KNN 0.556 ±.071 (0.153)	RFC 0.347 ±.039 (-0.056)	KNN 0.583 ±.034 (0.18)	RFC 0.556 ±.052 (0.153)	RFC 0.542 ±.0 (0.139)	MLP 0.597 ±.039 (0.194)	**RFC** **0.611** **±.02** **(0.208)**	LOGIT 0.5 ±.034 (0.097)
adult	DTC 0.635 ±.006	SVM 0.836 ±.007 (0.201)	SVM 0.853 ±.002 (0.218)	**SVM** **0.903** **±.004** **(0.268)**	SVM 0.829 ±.005 (0.194)	RFC 0.645 ±.001 (0.01)	SVM 0.853 ±.005 (0.218)	SVM 0.832 ±.006 (0.197)	SVM 0.816 ±.01 (0.181)	SVM 0.861 ±.005 (0.226)	SVM 0.804 ±.015 (0.169)	LOGIT 0.705 ±.009 (0.07)
yeast3	MLP 0.755 ±.067	LOGIT 0.871 ±.035 (0.116)	KNN 0.884 ±.038 (0.129)	SVM 0.952 ±.01 (0.197)	KNN 0.884 ±.067 (0.129)	SVM 0.864 ±.035 (0.109)	KNN 0.864 ±.038 (0.109)	SVM 0.912 ±.035 (0.157)	SVM 0.837 ±.029 (0.082)	SVM 0.952 ±.025 (0.197)	**LOGIT** **0.959** **±.017** **(0.204)**	SVM 0.932 ±.025 (0.177)
abalone9-18	DTC 0.436 ±.036	SVM 0.795 ±.131 (0.359)	SVM 0.846 ±.063 (0.41)	SVM 0.795 ±.073 (0.359)	LOGIT 0.846 ±.109 (0.41)	nan	LOGIT 0.846 ±.063 (0.41)	SVM 0.615 ±.218 (0.179)	SVM 0.615 ±.063 (0.179)	**SVM** **0.897** **±.073** **(0.461)**	MLP 0.744 ±.145 (0.308)	LOGIT 0.744 ±.036 (0.308)
winequality-red-4	DTC 0.104 ±.029	SVM 0.708 ±.106 (0.604)	**SVM** **0.729** **±.106** **(0.625)**	SVM 0.708 ±.059 (0.604)	**SVM** **0.729** **±.106** **(0.625)**	SVM 0.5 ±.051 (0.396)	**SVM** **0.729** **±.106** **(0.625)**	LOGIT 0.542 ±.029 (0.438)	SVM 0.479 ±.059 (0.375)	SVM 0.667 ±.029 (0.563)	LOGIT 0.646 ±.059 (0.542)	SVM 0.5 ±.051 (0.396)
mammography	DTC 0.618 ±.019	LOGIT 0.89 ±.033 (0.272)	LOGIT 0.886 ±.012 (0.268)	**SVM** **0.921** **±.011** **(0.303)**	KNN 0.851 ±.016 (0.233)	SVM 0.754 ±.041 (0.136)	LOGIT 0.873 ±.016 (0.255)	MLP 0.882 ±.019 (0.264)	SVM 0.833 ±.012 (0.215)	LOGIT 0.886 ±.016 (0.268)	LOGIT 0.882 ±.021 (0.264)	SVM 0.851 ±.006 (0.233)
abalone-20-vs-8-9-10	MLP 0.292 ±.059	**LOGIT** **0.875** **±.102** **(0.583)**	LOGIT 0.75 ±.102 (0.458)	SVM 0.75 ±.102 (0.458)	LOGIT 0.75 ±.102 (0.458)	LOGIT 0.625 ±.0 (0.333)	LOGIT 0.792 ±.059 (0.5)	MLP 0.5 ±.0 (0.208)	DTC 0.417 ±.059 (0.125)	LOGIT 0.667 ±.212 (0.375)	SVM 0.833 ±.059 (0.541)	LOGIT 0.667 ±.059 (0.375)

Table B.9 Recall without Classifiers HPs Tuning: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

dataset	none	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
pima	DTC 0.593 ±.08	DTC 0.708 ±.084 (0.115)	DTC 0.749 ±.091 (0.156)	**SVM 0.774 ±.035 (0.181)**	DTC 0.745 ±.078 (0.152)	RFC 0.663 ±.006 (0.07)	LOGIT 0.733 ±.021 (0.14)	LOGIT 0.741 ±.017 (0.148)	LOGIT 0.704 ±.0 (0.111)	LOGIT 0.749 ±.056 (0.156)	MLP 0.724 ±.032 (0.131)	SVM 0.733 ±.032 (0.14)
german	MLP 0.544 ±.031	SVM 0.615 ±.014 (0.071)	SVM 0.696 ±.021 (0.152)	**SVM 0.77 ±.014 (0.226)**	SVM 0.674 ±.014 (0.13)	LOGIT 0.489 ±.0 (-0.055)	LOGIT 0.711 ±.027 (0.167)	SVM 0.652 ±.05 (0.108)	LOGIT 0.615 ±.014 (0.071)	SVM 0.693 ±.01 (0.149)	LOGIT 0.593 ±.055 (0.049)	MLP 0.533 ±.018 (-0.011)
haberman	DTC 0.403 ±.052	DTC 0.486 ±.104 (0.083)	DTC 0.569 ±.071 (0.166)	DTC 0.556 ±.086 (0.153)	DTC 0.583 ±.034 (0.18)	LOGIT 0.333 ±.0 (-0.07)	KNN 0.514 ±.109 (0.111)	RFC 0.542 ±.102 (0.139)	DTC 0.542 ±.0 (0.139)	MLP 0.597 ±.071 (0.194)	**DTC 0.639 ±.039 (0.236)**	SVM 0.486 ±.052 (0.083)
adult	RFC 0.632 ±.001	SVM 0.836 ±.007 (0.204)	SVM 0.853 ±.002 (0.221)	**LOGIT 0.901 ±.002 (0.269)**	SVM 0.842 ±.005 (0.21)	LOGIT 0.644 ±.013 (0.012)	SVM 0.853 ±.005 (0.221)	SVM 0.845 ±.012 (0.213)	SVM 0.82 ±.008 (0.188)	SVM 0.858 ±.004 (0.226)	DTC 0.865 ±.03 (0.233)	LOGIT 0.655 ±.027 (0.023)
yeast3	DTC 0.816 ±.093	MLP 0.871 ±.035 (0.055)	SVM 0.891 ±.042 (0.075)	**SVM 0.966 ±.019 (0.15)**	DTC 0.878 ±.05 (0.062)	SVM 0.898 ±.029 (0.082)	SVM 0.905 ±.048 (0.089)	SVM 0.912 ±.035 (0.096)	SVM 0.844 ±.038 (0.028)	SVM 0.952 ±.025 (0.136)	LOGIT 0.959 ±.017 (0.143)	LOGIT 0.932 ±.025 (0.116)
abalone9-18	LOGIT 0.538 ±.126	LOGIT 0.795 ±.131 (0.257)	SVM 0.846 ±.063 (0.308)	SVM 0.821 ±.096 (0.283)	SVM 0.795 ±.036 (0.257)	nan	**SVM 0.872 ±.073 (0.334)**	LOGIT 0.615 ±.218 (0.077)	SVM 0.615 ±.063 (0.077)	**LOGIT 0.872 ±.096 (0.334)**	MLP 0.718 ±.131 (0.18)	LOGIT 0.513 ±.131 (-0.025)
winequality-red-4	DTC 0.083 ±.029	LOGIT 0.708 ±.106 (0.625)	**SVM 0.792 ±.059 (0.709)**	SVM 0.688 ±.051 (0.605)	SVM 0.771 ±.078 (0.688)	LOGIT 0.5 ±.051 (0.417)	SVM 0.771 ±.078 (0.688)	LOGIT 0.542 ±.029 (0.459)	SVM 0.458 ±.078 (0.375)	LOGIT 0.667 ±.029 (0.584)	LOGIT 0.646 ±.059 (0.563)	SVM 0.479 ±.059 (0.396)
mammography	KNN 0.588 ±.038	LOGIT 0.89 ±.033 (0.302)	LOGIT 0.882 ±.011 (0.294)	**LOGIT 0.921 ±.011 (0.333)**	KNN 0.855 ±.019 (0.267)	SVM 0.772 ±.038 (0.184)	LOGIT 0.873 ±.016 (0.285)	MLP 0.886 ±.022 (0.298)	MLP 0.82 ±.038 (0.232)	LOGIT 0.886 ±.016 (0.298)	LOGIT 0.882 ±.021 (0.294)	LOGIT 0.851 ±.006 (0.263)
abalone-20-vs-8-9-10	MLP 0.292 ±.059	**SVM 0.875 ±.102 (0.583)**	SVM 0.75 ±.0 (0.458)	SVM 0.75 ±.102 (0.458)	SVM 0.792 ±.059 (0.5)	LOGIT 0.5 ±.0 (0.208)	SVM 0.792 ±.059 (0.5)	MLP 0.5 ±.0 (0.208)	MLP 0.333 ±.059 (0.041)	SVM 0.667 ±.295 (0.375)	LOGIT 0.833 ±.059 (0.541)	MLP 0.375 ±.102 (0.083)

Table B.10 Recall with Classifiers HPs tuning: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

results	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
better	9	9	9	9	6	9	9	9	9	9	9
same	0	0	0	0	0	0	0	0	0	0	0
worse	0	0	0	0	2	0	0	0	0	0	0
avg diff [%]	25.92	28.51	30.38	27.99	11.94	28.88	20.62	16.42	28.8	27.16	19.82
avg std	0.025	0.016	0.002	0.026	-0.012	0.006	0.013	-0.007	0.018	0.011	-0.005

Table B.11 Recall without Classifiers HPs Tuning Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse result than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

results	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
better	9	9	9	9	6	9	9	9	9	9	7
same	0	0	0	0	0	0	0	0	0	0	0
worse	0	0	0	0	2	0	0	0	0	0	2
avg diff [%]	25.5	28.21	29.53	27.18	10.6	28.17	19.4	14.02	27.24	26.33	11.87
avg std	0.012	-0.017	-0.01	-0.015	-0.031	-0.008	-0.003	-0.023	0.01	-0.007	-0.006

Table B.12 Recall with Classifiers HPs Tuning Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse result than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

dataset	none	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
pima	MLP 0.738 ±.021	MLP 0.733 ±.034 (-0.005)	SVM 0.758 ±.026 (0.02)	LOGIT 0.755 ±.043 (0.017)	**MLP 0.761 ±.03 (0.023)**	MLP 0.74 ±.026 (0.002)	SVM 0.75 ±.036 (0.012)	LOGIT 0.737 ±.037 (-0.001)	LOGIT 0.75 ±.027 (0.012)	SVM 0.747 ±.049 (0.009)	RFC 0.758 ±.034 (0.02)	MLP 0.754 ±.033 (0.016)
german	SVM 0.662 ±.014	LOGIT 0.691 ±.021 (0.029)	SVM 0.706 ±.022 (0.044)	**SVM 0.723 ±.017 (0.061)**	LOGIT 0.699 ±.028 (0.037)	LOGIT 0.644 ±.0 (-0.018)	LOGIT 0.712 ±.031 (0.05)	RFC 0.688 ±.03 (0.026)	LOGIT 0.696 ±.012 (0.034)	SVM 0.709 ±.022 (0.047)	LOGIT 0.68 ±.018 (0.018)	LOGIT 0.674 ±.016 (0.012)
haberman	RFC 0.59 ±.037	RFC 0.624 ±.069 (0.034)	MLP 0.66 ±.024 (0.07)	MLP 0.653 ±.051 (0.063)	**MLP 0.664 ±.05 (0.074)**	LOGIT 0.596 ±.022 (0.006)	MLP 0.662 ±.023 (0.072)	MLP 0.625 ±.008 (0.035)	RFC 0.58 ±.0 (-0.01)	MLP 0.637 ±.006 (0.047)	RFC 0.652 ±.055 (0.062)	MLP 0.635 ±.046 (0.045)
adult	RFC 0.777 ±.001	LOGIT 0.814 ±.002 (0.037)	**LOGIT 0.822 ±.001 (0.045)**	LOGIT 0.816 ±.001 (0.039)	SVM 0.82 ±.002 (0.043)	RFC 0.781 ±.001 (0.004)	**LOGIT 0.822 ±.001 (0.045)**	LOGIT 0.809 ±.007 (0.032)	LOGIT 0.802 ±.003 (0.025)	LOGIT 0.819 ±.001 (0.042)	LOGIT 0.81 ±.002 (0.033)	LOGIT 0.788 ±.004 (0.011)
yeast3	MLP 0.867 ±.032	MLP 0.899 ±.02 (0.032)	KNN 0.901 ±.013 (0.034)	LOGIT 0.888 ±.011 (0.021)	**KNN 0.908 ±.03 (0.041)**	KNN 0.889 ±.019 (0.022)	MLP 0.895 ±.009 (0.028)	RFC 0.881 ±.036 (0.014)	SVM 0.885 ±.009 (0.018)	LOGIT 0.888 ±.004 (0.021)	MLP 0.885 ±.003 (0.018)	MLP 0.858 ±.046 (-0.009)
abalone9-18	SVM 0.714 ±.046	SVM 0.822 ±.066 (0.108)	**SVM 0.884 ±.035 (0.17)**	SVM 0.852 ±.045 (0.138)	LOGIT 0.871 ±.06 (0.157)	nan	SVM 0.88 ±.034 (0.166)	SVM 0.781 ±.112 (0.067)	SVM 0.78 ±.031 (0.066)	SVM 0.775 ±.029 (0.061)	MLP 0.832 ±.066 (0.118)	LOGIT 0.836 ±.016 (0.122)
winequality-red-4	DTC 0.53 ±.018	LOGIT 0.733 ±.058 (0.203)	LOGIT 0.728 ±.041 (0.198)	**SVM 0.761 ±.04 (0.231)**	LOGIT 0.733 ±.037 (0.203)	LOGIT 0.691 ±.038 (0.161)	LOGIT 0.727 ±.043 (0.197)	LOGIT 0.682 ±.023 (0.152)	SVM 0.679 ±.021 (0.149)	LOGIT 0.661 ±.015 (0.131)	RFC 0.673 ±.049 (0.143)	SVM 0.622 ±.003 (0.092)
mammography	DTC 0.801 ±.009	MLP 0.901 ±.021 (0.1)	**MLP 0.916 ±.007 (0.115)**	MLP 0.903 ±.009 (0.102)	KNN 0.904 ±.011 (0.103)	SVM 0.863 ±.021 (0.062)	MLP 0.902 ±.009 (0.101)	MLP 0.893 ±.015 (0.092)	MLP 0.895 ±.016 (0.094)	LOGIT 0.871 ±.01 (0.07)	KNN 0.848 ±.019 (0.047)	LOGIT 0.83 ±.01 (0.029)
abalone-20-vs-8-9-10	MLP 0.645 ±.03	**LOGIT 0.894 ±.06 (0.249)**	LOGIT 0.851 ±.05 (0.206)	SVM 0.858 ±.053 (0.213)	LOGIT 0.851 ±.049 (0.206)	LOGIT 0.803 ±.0 (0.158)	LOGIT 0.865 ±.028 (0.22)	MLP 0.74 ±.004 (0.095)	DTC 0.697 ±.03 (0.052)	LOGIT 0.681 ±.103 (0.036)	SVM 0.823 ±.03 (0.178)	LOGIT 0.813 ±.033 (0.168)

Table B.13 AUC without Classifiers HPs Tuning: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

dataset	none	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
pima	RFC 0.722 ±.023	MLP 0.724 ±.013 (0.002)	LOGIT 0.754 ±.037 (0.032)	**LOGIT 0.756 ±.044 (0.034)**	RFC 0.753 ±.029 (0.031)	RFC 0.743 ±.024 (0.021)	SVM 0.751 ±.035 (0.029)	SVM 0.738 ±.037 (0.016)	LOGIT 0.75 ±.027 (0.028)	LOGIT 0.744 ±.05 (0.022)	MLP 0.742 ±.016 (0.02)	LOGIT 0.749 ±.041 (0.027)
german	MLP 0.688 ±.014	SVM 0.692 ±.02 (0.004)	SVM 0.704 ±.019 (0.016)	**SVM 0.724 ±.015 (0.036)**	LOGIT 0.698 ±.027 (0.01)	LOGIT 0.642 ±.0 (-0.046)	LOGIT 0.713 ±.031 (0.025)	RFC 0.696 ±.027 (0.008)	LOGIT 0.696 ±.012 (0.008)	SVM 0.707 ±.015 (0.019)	MLP 0.678 ±.025 (-0.01)	LOGIT 0.672 ±.014 (-0.016)
haberman	RFC 0.593 ±.032	MLP 0.627 ±.026 (0.034)	MLP 0.627 ±.06 (0.034)	MLP 0.633 ±.064 (0.04)	MLP 0.655 ±.063 (0.062)	LOGIT 0.606 ±.02 (0.013)	MLP 0.618 ±.052 (0.025)	RFC 0.631 ±.008 (0.038)	DTC 0.58 ±.0 (-0.013)	RFC 0.648 ±.037 (0.055)	**RFC 0.667 ±.036 (0.074)**	MLP 0.64 ±.04 (0.047)
adult	RFC 0.778 ±.001	LOGIT 0.814 ±.002 (0.036)	LOGIT 0.821 ±.001 (0.043)	LOGIT 0.816 ±.001 (0.038)	SVM 0.821 ±.002 (0.043)	RFC 0.781 ±.001 (0.003)	**LOGIT 0.822 ±.001 (0.044)**	SVM 0.81 ±.006 (0.032)	LOGIT 0.802 ±.003 (0.024)	LOGIT 0.819 ±.001 (0.041)	SVM 0.81 ±.002 (0.032)	LOGIT 0.782 ±.008 (0.004)
yeast3	DTC 0.89 ±.036	MLP 0.906 ±.013 (0.016)	KNN 0.894 ±.025 (0.004)	LOGIT 0.887 ±.01 (-0.003)	**DTC 0.909 ±.022 (0.019)**	SVM 0.898 ±.016 (0.008)	SVM 0.896 ±.021 (0.006)	DTC 0.894 ±.031 (0.004)	LOGIT 0.887 ±.007 (-0.003)	DTC 0.905 ±.036 (0.015)	DTC 0.905 ±.03 (0.015)	DTC 0.893 ±.031 (0.003)
abalone9-18	LOGIT 0.764 ±.061	LOGIT 0.818 ±.063 (0.054)	LOGIT 0.882 ±.035 (0.118)	SVM 0.868 ±.053 (0.104)	LOGIT 0.859 ±.022 (0.095)	nan	**SVM 0.889 ±.038 (0.125)**	LOGIT 0.775 ±.112 (0.011)	SVM 0.78 ±.031 (0.016)	LOGIT 0.763 ±.038 (-0.001)	MLP 0.825 ±.059 (0.061)	LOGIT 0.751 ±.064 (-0.013)
winequality-red-4	KNN 0.536 ±.015	LOGIT 0.733 ±.058 (0.197)	SVM 0.749 ±.015 (0.213)	**LOGIT 0.754 ±.038 (0.218)**	SVM 0.737 ±.019 (0.201)	LOGIT 0.692 ±.039 (0.156)	SVM 0.741 ±.033 (0.205)	LOGIT 0.682 ±.023 (0.146)	SVM 0.678 ±.032 (0.142)	LOGIT 0.661 ±.014 (0.125)	MLP 0.673 ±.055 (0.137)	LOGIT 0.615 ±.015 (0.079)
mammography	KNN 0.791 ±.019	MLP 0.899 ±.011 (0.108)	**MLP 0.911 ±.008 (0.12)**	MLP 0.898 ±.007 (0.107)	KNN 0.906 ±.012 (0.115)	SVM 0.873 ±.02 (0.082)	MLP 0.907 ±.006 (0.116)	MLP 0.895 ±.02 (0.104)	MLP 0.895 ±.017 (0.104)	LOGIT 0.871 ±.01 (0.08)	LOGIT 0.847 ±.024 (0.056)	LOGIT 0.83 ±.01 (0.039)
abalone-20-vs-8-9-10	LOGIT 0.645 ±.029	**SVM 0.901 ±.059 (0.256)**	SVM 0.85 ±.0 (0.205)	SVM 0.857 ±.054 (0.212)	SVM 0.867 ±.027 (0.222)	LOGIT 0.74 ±.0 (0.095)	LOGIT 0.866 ±.028 (0.221)	MLP 0.739 ±.005 (0.094)	MLP 0.659 ±.024 (0.014)	LOGIT 0.683 ±.103 (0.038)	LOGIT 0.822 ±.031 (0.177)	MLP 0.686 ±.05 (0.041)

Table B.14 AUC with Classifiers HPs Tuning: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

results	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
better	8	9	9	9	7	9	8	8	9	9	8
same	0	0	0	0	0	0	0	0	0	0	0
worse	1	0	0	0	1	0	1	1	0	0	1
avg diff [%]	8.74	10.02	9.83	9.86	4.96	9.9	5.69	4.89	5.16	7.08	5.4
avg std	0.016	0.001	0.007	0.01	-0.004	0.001	0.007	-0.007	0.003	0.008	-0.0

Table B.15 AUC without Classifiers HPs Tuning Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse result than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

results	lit-gan (ours)	smote	borderline	poly	k-means	loras	ctgan	tvae	taei	ctab-gan	sm-gan
better	9	9	8	9	7	9	9	7	8	8	7
same	0	0	0	0	0	0	0	0	0	0	0
worse	0	0	1	0	1	0	0	2	1	1	2
avg diff [%]	7.86	8.72	8.73	8.87	4.15	8.84	5.03	3.56	4.38	6.24	2.34
avg std	0.004	-0.003	0.006	-0.001	-0.006	0.002	0.004	-0.009	0.008	0.005	0.005

Table B.16 AUC with Classifiers HPs Tuning Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse result than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

dataset	none	litgan-smote	litgan-poly	litgan-borderline
pima	MLP 0.738 ±.021	MLP 0.733 ±.034 (-0.005)	**LOGIT 0.743 ±.036 (0.005)**	MLP 0.737 ±.017 (-0.001)
german	SVM 0.662 ±.014	**LOGIT 0.691 ±.021 (0.029)**	SVM 0.69 ±.021 (0.028)	SVM 0.69 ±.032 (0.028)
haberman	RFC 0.59 ±.037	**RFC 0.624 ±.069 (0.034)**	RFC 0.551 ±.076 (-0.039)	MLP 0.59 ±.033 (0.0)
adult	RFC 0.777 ±.001	**LOGIT 0.814 ±.002 (0.037)**	LOGIT 0.811 ±.003 (0.034)	SVM 0.813 ±.002 (0.036)
yeast3	MLP 0.867 ±.032	**MLP 0.899 ±.02 (0.032)**	SVM 0.896 ±.018 (0.029)	SVM 0.882 ±.001 (0.015)
abalone9-18	SVM 0.714 ±.046	**SVM 0.822 ±.066 (0.108)**	SVM 0.766 ±.013 (0.052)	SVM 0.756 ±.059 (0.042)
winequality-red-4	DTC 0.53 ±.018	**LOGIT 0.733 ±.058 (0.203)**	LOGIT 0.709 ±.04 (0.179)	SVM 0.712 ±.042 (0.182)
mammography	DTC 0.801 ±.009	MLP 0.901 ±.021 (0.1)	**MLP 0.913 ±.011 (0.112)**	MLP 0.894 ±.01 (0.093)
abalone-20-vs-8-9-10	MLP 0.645 ±.03	**LOGIT 0.894 ±.06 (0.249)**	SVM 0.843 ±.044 (0.198)	MLP 0.645 ±.03 (0.0)

Table B.17 Balanced Accuracy of LIT-GAN Variants: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

results	litgan-smote	litgan-poly	litgan-borderline
better	8	8	6
same	0	0	2
worse	1	1	1
avg diff [%]	8.74	6.64	4.39
avg std	0.016	0.006	0.002

Table B.18 Balanced Accuracy of LIT-GAN Variants Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse result than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

dataset	none	litgan-smote	litgan-poly	litgan-borderline
pima	**MLP 0.71 ±.077**	MLP 0.641 ±.079 (-0.069)	MLP 0.661 ±.061 (-0.049)	RFC 0.665 ±.084 (-0.045)
german	**RFC 0.673 ±.02**	RFC 0.575 ±.01 (-0.098)	RFC 0.594 ±.031 (-0.079)	RFC 0.59 ±.017 (-0.083)
haberman	**SVM 0.72 ±.227**	MLP 0.494 ±.046 (-0.226)	RFC 0.365 ±.131 (-0.355)	MLP 0.441 ±.105 (-0.279)
adult	**LOGIT 0.74 ±.011**	RFC 0.704 ±.005 (-0.036)	RFC 0.706 ±.001 (-0.034)	RFC 0.708 ±.007 (-0.032)
yeast3	**KNN 0.827 ±.018**	RFC 0.79 ±.046 (-0.037)	RFC 0.791 ±.05 (-0.036)	RFC 0.8 ±.072 (-0.027)
abalone9-18	**LOGIT 1.0 ±.0**	MLP 0.535 ±.161 (-0.465)	KNN 0.537 ±.094 (-0.463)	MLP 0.769 ±.182 (-0.231)
winequality-red-4	**KNN 0.333 ±.471**	RFC 0.187 ±.035 (-0.146)	MLP 0.174 ±.048 (-0.159)	RFC 0.272 ±.097 (-0.061)
mammography	**SVM 0.875 ±.102**	RFC 0.558 ±.076 (-0.317)	RFC 0.502 ±.029 (-0.373)	RFC 0.607 ±.08 (-0.268)
abalone-20-vs-8-9-10	**SVM 0.778 ±.157**	MLP 0.505 ±.217 (-0.273)	MLP 0.277 ±.124 (-0.501)	MLP 0.639 ±.104 (-0.139)

■ **Table B.19** Precision of LIT-GAN Variants: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

results	litgan-smote	litgan-poly	litgan-borderline
better	0	0	0
same	0	0	0
worse	9	9	9
avg diff [%]	-18.52	-22.77	-12.94
avg std	-0.045	-0.057	-0.037

■ **Table B.20** Precision of LIT-GAN Variants Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse result than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

dataset	none	litgan-smote	litgan-poly	litgan-borderline
pima	DTC 0.63 ±.03	MLP 0.679 ±.03 (0.049)	**LOGIT 0.683 ±.023 (0.053)**	LOGIT 0.679 ±.017 (0.049)
german	DTC 0.507 ±.01	SVM 0.615 ±.014 (0.108)	**SVM 0.622 ±.042 (0.115)**	SVM 0.611 ±.033 (0.104)
haberman	DTC 0.403 ±.052	LOGIT 0.444 ±.071 (0.041)	**LOGIT 0.472 ±.071 (0.069)**	RFC 0.458 ±.148 (0.055)
adult	DTC 0.635 ±.006	**SVM 0.836 ±.007 (0.201)**	SVM 0.832 ±.005 (0.197)	SVM 0.835 ±.004 (0.2)
yeast3	MLP 0.755 ±.067	**LOGIT 0.871 ±.035 (0.116)**	LOGIT 0.85 ±.038 (0.095)	SVM 0.844 ±.025 (0.089)
abalone9-18	DTC 0.436 ±.036	**SVM 0.795 ±.131 (0.359)**	LOGIT 0.641 ±.036 (0.205)	SVM 0.538 ±.109 (0.102)
winequality-red-4	DTC 0.104 ±.029	**SVM 0.708 ±.106 (0.604)**	LOGIT 0.688 ±.102 (0.584)	SVM 0.562 ±.051 (0.458)
mammography	DTC 0.618 ±.019	**LOGIT 0.89 ±.033 (0.272)**	MLP 0.873 ±.033 (0.255)	SVM 0.846 ±.053 (0.228)
abalone-20-vs-8-9-10	MLP 0.292 ±.059	**LOGIT 0.875 ±.102 (0.583)**	LOGIT 0.792 ±.059 (0.5)	DTC 0.292 ±.059 (0.0)

Table B.21 Recall of LIT-GAN Variants: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

results	litgan-smote	litgan-poly	litgan-borderline
better	9	9	8
same	0	0	1
worse	0	0	0
avg diff [%]	25.92	23.03	14.28
avg std	0.025	0.011	0.021

Table B.22 Recallof LIT-GAN Variants Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse result than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

dataset	none	litgan-smote	litgan-poly	litgan-borderline
pima	MLP 0.738 ±.021	MLP 0.733 ±.034 (-0.005)	**LOGIT 0.743 ±.036 (0.005)**	MLP 0.737 ±.017 (-0.001)
german	SVM 0.662 ±.014	**LOGIT 0.691 ±.021 (0.029)**	SVM 0.69 ±.021 (0.028)	SVM 0.69 ±.032 (0.028)
haberman	RFC 0.59 ±.037	**RFC 0.624 ±.069 (0.034)**	RFC 0.551 ±.076 (-0.039)	MLP 0.59 ±.033 (0.0)
adult	RFC 0.777 ±.001	**LOGIT 0.814 ±.002 (0.037)**	LOGIT 0.811 ±.003 (0.034)	SVM 0.813 ±.002 (0.036)
yeast3	MLP 0.867 ±.032	**MLP 0.899 ±.02 (0.032)**	SVM 0.896 ±.018 (0.029)	SVM 0.882 ±.001 (0.015)
abalone9-18	SVM 0.714 ±.046	**SVM 0.822 ±.066 (0.108)**	SVM 0.766 ±.013 (0.052)	SVM 0.756 ±.059 (0.042)
winequality-red-4	DTC 0.53 ±.018	**LOGIT 0.733 ±.058 (0.203)**	LOGIT 0.709 ±.04 (0.179)	SVM 0.712 ±.042 (0.182)
mammography	DTC 0.801 ±.009	MLP 0.901 ±.021 (0.1)	**MLP 0.913 ±.011 (0.112)**	MLP 0.894 ±.01 (0.093)
abalone-20-vs-8-9-10	MLP 0.645 ±.03	**LOGIT 0.894 ±.06 (0.249)**	SVM 0.843 ±.044 (0.198)	MLP 0.645 ±.03 (0.0)

Table B.23 AUC of LIT-GAN Variants: Best performing classifier, mean value, standard deviation of the results, and difference from the results obtained on the original dataset. The best result is highlighted.

results	litgan-smote	litgan-poly	litgan-borderline
better	8	8	6
same	0	0	2
worse	1	1	1
avg diff [%]	8.74	6.64	4.39
avg std	0.016	0.006	0.002

Table B.24 AUC of LIT-GAN Variants Summary: Table shows how many times classification with the oversampled dataset for each technique gives better, the same, or worse result than classification on the original dataset. Avg diff and avg std are the average differences from means and standard deviations obtained on the original datasets. Avg diff is presented in percentages.

Appendix C
Hyperparameters of Classifiers

During the evaluation, hyperparameters of the classification models were tuned. The values for each model used in the grid search can be seen in the following tables.

hyperparameter	values
max_depth	3, 5, None
criterion	gini, entropy

Table C.1 Hyperparameters for Decision Tree Classifier

hyperparameter	values
activation	relu, logistic
max_iter	200, 400

Table C.2 Hyperparameters for MLP Classifier

hyperparameter	values
penalty	l1, l2
solver	lbfgs, liblinear
C	1, 10

Table C.3 Hyperparameters for Logistic Regression

hyperparameter	values
max_depth	3, 5, None
criterion	gini, entropy

Table C.4 Hyperparameters for Random Forest Classifier

hyperparameter	values
penalty	l1, l2
C	1, 10
loss	hinge, squared_hinge

■ **Table C.5** Hyperparameters for Linear SVM

hyperparameter	values
n_neighbors	3, 5, 7
weights	uniform, distance

■ **Table C.6** Hyperparameters for k Neighbors Classifier