FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Generative methods applicable for data anonymization and test data creation in the banking industry
Student:	Jan Jeníček
Supervisor:	Mgr. Ivo Kylar
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2021/2022

Instructions

In the banking industry, there are many scenarios when it is advantageous to generate data that seem to be real. It can be used for both model testing and as a way of anonymization in case the data are sensitive.

1) Introduce and describe generative methods suitable for generating artificial customer data derived from a production data set.
2) Choose the best method and explain your choice in detail.
3) Implement the chosen method and evaluate its performance.
4) Suggest future improvements following your implementation.

Electronically approved by Ing. Karel Klouda, Ph.D. on 4 January 2021 in Prague.

Bachelor thesis

GENERATIVE METHODS APPLICABLE FOR DATA ANONYMIZATION AND TEST DATA CREATION IN THE BANKING INDUSTRY

Jan Jeníček

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Mgr. Ivo Kylar
May 13, 2021

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on May 10, 2021 .....

Acknowledgements

I would like to thank Komerční Banka for the oppurtunity to cooperate on the bachelor thesis. Namely, I would like to thank Mgr. Ivo Kylar and RNDr. Tomáš Pazák, Ph.D. for their guidance and Mrg. Matěj Stránský for the initial organization.

Abstract

Production tabular data are often extremely sensitive and thus difficult to manipulate. My thesis surveys generative machine learning methods and their ability to learn data distributions. This ability is used to generate new synthetic data that can completely replace the original data where it is needed. The thesis provides a theoretical basis with a strong focus on two significant deep learning approaches, VAE and GAN. Existing GAN modification for tabular data, CTGAN, is implemented and trained with banking dataset as a proof of concept. Results from different metrics show that the chosen anonymization approach of banking data might be very successful and steps following this thesis are suggested. The thesis was written in collaboration with Komerční Banka.

Keywords generative methods, banking data, generative adversarial networks, tabular data, CTGAN, anonymization, data augmentation

Abstrakt

Produkční data často obsahují velmi citlivé informace a je složité s nimi manipulovat. Má práce zkoumá generativní metody strojového učení a jejich schopnost naučit se distribuci dat. Tato schopnost je použita pro generování syntetických dat, která mohou v případě potřeby kompletně nahradit ty reálná. Práce nejprve prezentuje teoretický základ a věnuje se do hloubky dvěma metodám hlubokého učení, VAE a GAN. Existující modifikace GAN pro tabulková data, CTGAN, je implementována a na bankovních datech je natrénován model. Výsledky vycházející z různých metrik ukazují, že tento způsob anonymizace bankovních dat může být velmi úspěšný a jsou navrženy další kroky, které mohou zakládat na této práci. Práce je tvořena ve spolupráci s Komerční Bankou.

Klíčová slova generativní metody, bankovní data, generativní adversariální sítě, tabulková data, CTGAN, anonymizace, augmentace dat

Contents

List of Figures

List of Tables

Abbreviations

GDPR	General Data Protection Regulation
VAE	Variational Autoencoder
GAN	Generative Adversarial Networks
MLE	Maximum Likelihood Estimation
WGAN	Wasserstein Generative Adversarial Networks
GP	Gradient Penalty
CTGAN	Conditional Tabular Generative Adversarial Networks
ML	Machine Learning
CDF	Cumulative Distribution Function
SDV	Synthetic Data Vault
NVC	Null Values Constraints

Chapter 1

Introduction

Motivation

Many workflows heavily rely on good-quality data. Although the pace of producing new data is faster than ever in history, it might be very complicated to access the data. Making copies of original data can lead to safety issues and increases the risk of data breaches. Additionally, regulations such as GDPR further limit the possibility to spread data containing sensitive personal information. All companies which collect customer data to improve their services are affected because they usually need the data for their development and testing routines. This often requires moving and copying the data a lot, and therefore indirectly threatens the original data source. By all means, the banking industry is a representative area considering this issue due to strong regulations and high sensitivity of financial data.

Data masking solves the issue by replacing sensitive data with artificial values. However, the approach faces many challenges. Masked data must retain all real data aspects and when used in a certain process, the result must be consistently similar. Also, the algorithm must not be vulnerable to reverse engineering as it would make the masking useless. There are many techniques which can be used to replace certain values in the data. With that being said, they are formed by complex rules which respect certain constraints and therefore can be very limited.

On the other hand, generative models aim to learn data distribution so they can naturally sample new data from it without defining complex rules. In theory, this synthetic data should not differ from the real data as they come from the same probabilistic distribution. When using a generative model, no original features are taken from the real dataset, and instead, a completely synthetic one is generated. This prevents everyone who is working with the synthetic data from knowing anything about the original data.

Also, the generative model can be useful outside of the data masking scope. When performing a task, which requires an extensive amount of data, we can use the synthetic data generated by the model alongside the real ones and thus increase the amount of data available for the task. This technique is often called data augmentation.

Objectives

There has been a serious focus on generative methods over the past decade. A lot of papers and new algorithms for synthetic data generation have been published and the field is evolving very fast.

The thesis surveys generative methods which can be used to anonymize and augment data in Komerční Banka. Its goal is to thoroughly review generative machine learning approach with respect to its ability to create synthetic data. State-of-the-art methods which are applicable for tabular banking data should be analyzed and one method selected based on the analysis and its suitability for the bank's environment. The method has to be implemented and trained with a production dataset provided by Komerční Banka. The model then has to be evaluated using different techniques. Lastly, further changes and improvements should be presented.

Thesis Structure

Including this chapter, the thesis consists of 7 chapters. The second chapter introduces synthetic data and their usage. The third chapter describes generative models and presents the most important representatives. The fourth chapter presents specifics for tabular data and algorithms designed for them and thoroughly explains the selected method. Implementation details are presented in the fifth chapter and the model is evaluated in the sixth chapter. The last chapter discusses future improvements to the implementation.

Chapter 2

Synthetic Data

Unlike the data collected in real-world situations, synthetic data are completely generated by a program. They can be utilized in various manners and help us overcome many challenges which come with the real-world data.

One of the most significant features is the simplicity of obtaining such data. Once we know how to generate new instances, we can create large synthetic datasets with little effort. This is especially useful in machine learning as many algorithms require an extensive amount of training data to perform as intended. Additionally, the algorithms can produce perfectly clean data which do not need to be preprocessed for further use. There are no missing or ambiguous values in the dataset unless we want them to be there. Also, the fact that the data do not come from a real-world source makes it much safer to manipulate, copy and move the data as it is less likely for them to be misused by a third party.

The biggest challenge, in general, is to perfectly mimic the data coming from the real world. In a perfect scenario, the synthetic dataset should not be recognizable from the real world one concerning a task performed. Also, the importance of the data realness usually differs, meaning that there are many methods to generate the data varying in complexity. The simplest methods draw the data from a predefined probabilistic distribution, such as Gaussian, and therefore need no additional training [1]. Although they are widely used for simple tasks and demonstrative purposes, they lack the ability to provide any useful information, so they cannot be used in more complex scenarios. When trying to generate realistic data, a generative machine learning model has to be trained using a real dataset. The model learns the real data distribution and then samples new data from it, theoretically preserving the real-world data characteristics. Generative models can be used to solve various machine learning problems, yet this thesis further discusses their application to data augmentation and anonymization.

Data Augmentation

Data augmentation techniques aim to increase the number of samples in our dataset. It can be advantageous for multiple reasons, such as prevention of overfitting and model stability improving, and therefore boost the overall performance of our model. Although the algorithms used depend on many dataset specifics, generative models, in general, are being used for data augmentation across all machine learning fields when there is a need of good-quality data.

The augmentation can also be advantageous when we have enough data at the first glance. To set an example, many machine learning models behave differently when they are provided highly imbalanced data. Highly imbalanced data occur a lot in real world, especially in datasets related to anomalies such as fraud detection. To rebalance the dataset, we can augment it first and then down-sample it back to its original size, preserving all minority class data points and thus rebalancing the dataset.

Data Anonymization

Data anonymization can generally be defined as a process of data transformation which aims to make an identification of the former data point impossible. In other words, the anonymization should be irreversible and it should protect the information we want to anonymize from any other party. This thesis aims to accomplish that by replacing all real data with fake samples generated by a generative model. The process is naturally irreversible as the anonymized data points do not come from the original ones and with correct data preprocessing, no sensitive information is projected into the new dataset.

It is often not easy to achieve full anonymization of a dataset. Even when all sensitive information is covered or changed, a set of values which appears together can still define a very specific situation. In such case, it would still be possible to identify original values with a prior domain knowledge. As a result of that, more complex anonymization technique is often required.

Chapter 3

Generative Methods

There are two different machine learning approaches to a classification task using statistics. Consider a general machine learning classification task, a feature vector x and a target variable y.

- *Discriminative methods* aim to learn the conditional probability distribution $P(y \mid x)$. In other words, they find boundaries in the data vector space for all values of the target variable with respect to the training data.
- *Generative methods* aim to learn the joint probability distribution $P(x,\ y)$, which they use to infer the conditional probability using the Bayes theorem.

Discriminative models used to be superior and widely used until the last decade. Although this has mostly changed thanks to various deep generative models which were introduced recently (3.1), there are both advantages and disadvantages of each approach when it comes to the classification task and the comparison can get very complicated. However, the approaches differ a lot considering new data generation. Contrarily to generative methods, discriminative methods do not recognize the underlying distribution of the data and therefore cannot be used for the data generation task when we train them. This is why they are not discussed any further in the thesis.

This thesis focuses on deep models as they are considered to be one of the most promising machine learning areas and majority of the state-of-the-art algorithms use deep learning techniques for data generation.

3.1 Deep Generative Methods

Deep generative models combine the ability to learn data distribution with capabilities of neural networks and deep learning techniques. The universal approximation theorem, first proven in [2], tells us that we can approximate any given function with a neural network, so we are theoretically able to approximate any data distribution. This makes deep models generally very powerful when it comes to solving complex generative problems with large inputs, including reconstruction or generation of high resolution images [3].

Additionally, deep generative algorithms are usually unsupervised. While supervised methods require labelling the training data in order to learn, unsupervised methods do not need any guidance. This means we need nothing but a dataset to train such model and it expands the suitability of deep generative models for synthetic data generation even further.

In the following part, two key deep generative approaches, variational autoencoders and generative adversarial networks, will be described. Their fundamental components will be explained, showing their significant characteristics and why they are so popular and widely used.

3.2 VAE

This section describes variational autoencoder (VAE), originally introduced in 2013 [4], and its base components. First, the vanilla autoencoder architecture is briefly outlined. After that, its limits are discussed, presenting all improvements to the VAE and its data-generative features.

Autoencoder

Autoencoder is an unsupervised neural architecture which aims to learn a representation of a given dataset. The architecture is made of two neural networks, an encoder E and a decoder D. In each iteration of the learning process, E takes a data point x and encodes it into a new latent representation z. D then decodes z back to the original space, creating new data point x_d. Decoded point x_d is then compared to x and the error between the points determined by a loss function $L(x, x_d)$ is backpropagated through both neural networks, forcing the output to be as similar to the original input as possible. The idea behind the algorithm is displayed in figure 3.1.

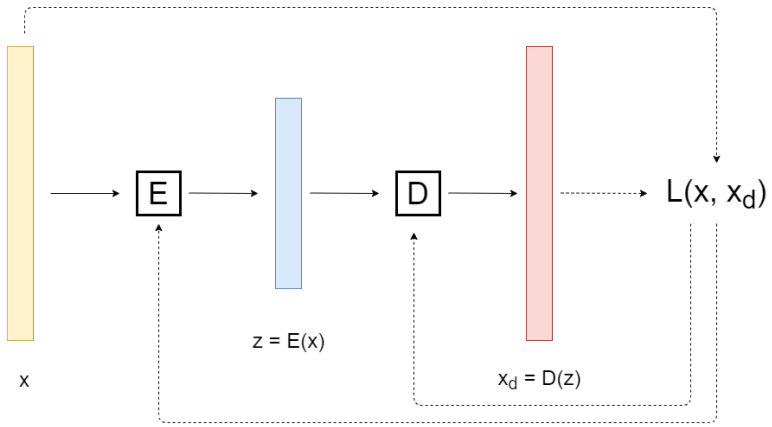


Figure 3.1 Architecture of autoencoder

In most cases, a dimension of the latent space Z is lower than a dimension of the input space X. Thus, the input is compressed by the encoder, potentially loosing some of the initial information. As a result of that, the latent representation is forced

to keep the most important features as the decoder tries to recreate the original input from it. Finding such encoding of reduced dimension can be extremely useful in many scenarios, ranging from noise filtering to dimensionality reduction. In that case for example, the autoencoder works similarly to a principal component analysis in linear algebra, being even more powerful thanks to the ability of neural networks to apply non-linear transformations.

Although the autoencoder is good at finding different data representation, it is nearly impossible to generate good quality data without any algorithm modification. The latent space tends to be highly irregular, making it difficult to create plausible data by sampling latent points sampling. In short, the space irregularity is a scenario when two similar points are encoded into very distant latent representations. The original data structure becomes corrupted and the latent space becomes seemingly randomly organized. When this happens, there is no guarantee that a point sampled from the latent space by averaging two real encodings would give us anything close to a mixture of the original data points. Because of that, a latent space regularization technique is needed in order to make a data generation possible.

Latent Space Regularization

To generate completely new data samples, VAE uses slightly different training technique in order to ensure the latent space regularity. Instead of encoding the input as a point in the latent space, it is encoded as a probabilistic distribution over the latent space. Normal distribution is used, meaning that we can naturally encode the input as a pair of variables defining a normal distribution, a mean and a variance. Then, a point is sampled from the distribution and the algorithm follows the vanilla autoencoder training algorithm. The sampled latent point is decoded and the error returned by a modified loss function is backpropagated through the autoencoder network.

In theory, this change to the training process helps to regularize the latent space as the decoder input can be sampled anywhere from a distribution returned by the encoder, meaning the distributions now have to be organized as they might overlap. However, the technique alone does not work in practice, because the relation between a mean and a variance of each distribution over the latent space is not restricted with respect to each other [5]. As a result of that, all distributions can have extremely low variance considering the distances among them, behaving almost the same as a vanilla autoencoder which works with simple points.

To prevent this scenario, the covariance matrix of each latent distribution is forced to be as close to the identity as possible, presuming no correlation between any of the latent vector elements, while keeping the means of the distributions as centered as possible. This is achieved by a change to the loss function, which penalizes the autoencoder not only when the reconstructed output differs from the input, but also when the latent space distribution diverges from the standard normal one. Figure 3.2 shows the modified algorithm.

Simply put, the encoded distributions are squeezed close to each other and they aim to maintain a shape of the standard normal distribution by encoding the input into multivariate normal distributions instead of single points. Further, the latent space is forced to be more regular, because the encoded representations of the inputs overlap and thus have to be systematically organized with respect to each other. Once the algorithm is trained, synthetic data can be generated by sampling from a normal distribution over the latent space.

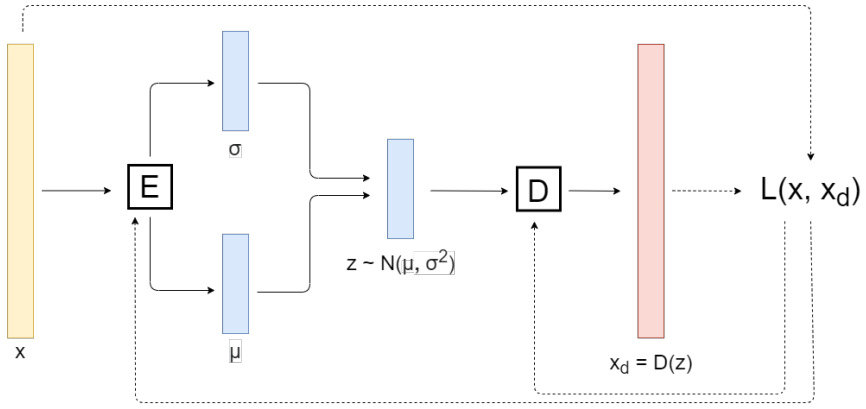


Figure 3.2 Architecture of VAE

3.3 GAN

Generative adversarial networks (GAN) is an approach first introduced in 2014 [6] which caused tremendous surge in machine learning research related to deep generative models. Although many improvements to the original algorithm have been proposed [7, 8], this section will outline the components of the original vanilla GAN as all improved algorithms are naturally based on it.

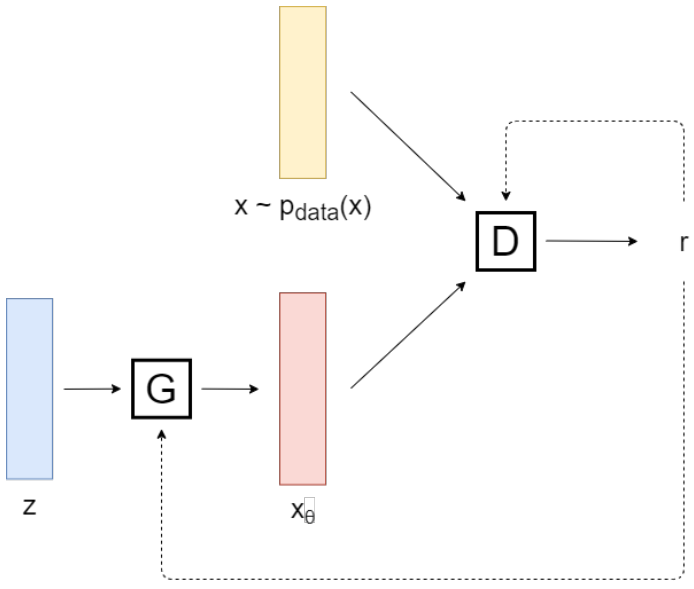


Figure 3.3 Architecture of GAN

The architecture of GAN can be seen in figure 3.3. The algorithm consists of two neural networks, a generator G and a discriminator D, which play a two-player minimax game against each other. G takes a random noise sampled from a prior as an input and transforms it into a fake data point x_θ. D is given both real and fake samples with a goal to distinguish them. The realness of an input is indicated by a scalar r ranging from 0 to 1. During the training process, D tries to maximize the chance of classifying its input correctly while G aims to confuse D, aiming to minimize the chance. The minimax game value function $V(D, G)$ can formally be defined as:

$$\min_G \max_D V(D,G) = \mathbf{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbf{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{3.1}$$

where z is an input noise sampled from a prior $p_z(z)$ and x is a real data point coming from the data distribution $x \sim p_{data}(x)$.

As proven in the original article [6], a unique perfect solution to the game exists, with G successfully learning the underlying data distribution and D being unable to detect artificial data points, returning $\frac{1}{2}$ for every input. Both D and G are multilayer perceptron neural networks, which means they can approximate any function or distribution and, in theory, can converge to this solution via backpropagation. Loss functions for both D (equation 3.2) and G (equation 3.3) can be directly inferred from the value function.

There are two differences in the loss functions. First, both functions differ in sign as the networks play a minimax game and what one minimizes, the other maximizes. Additionally, part related to the classification of real images is omitted from the generator loss, because they are not related to it by any means.

$$L_D = -[\log(D(x)) + \log(1 - D(G(z)))] \tag{3.2}$$

$$L_G = \log(1 - D(G(z))) \tag{3.3}$$

After the training process is complete, it is straightforward to generate synthetic data with GAN. Since the generator was trained to create as real data points as possible, it can be used the same way to generate completely new data from a random noise input. The discriminator network is contrarily only used for the generator training and it does not participate in the data generation process.

3.4 VAE And GAN Comparison

This section highlights advantages and disadvantages of VAEs and GANs and their key differences are analyzed. Although newer methods might solve some of the disadvantages and their architecture might be different, the vanilla methods, presented in 3.2 and 3.3, will be compared as their characteristics discussed are highly important for every derived algorithm's usage and they are essential when choosing an algorithm to be used.

As already mentioned, all generative models are used to learn the data distribution. The most significant difference among all of them is the training procedure and the objective function, because they together define how the model aims to converge towards that distribution. This applies to VAEs and GANs too and the difference in the training has several consequences.

For better interpretation of the difference, the maximum likelihood estimation method (MLE) and the likelihood (equation 3.4) need to be defined first. A generative model is defined by a set of parameters θ, for instance, the weights of a neural network. $\mathcal{L}(X, \theta)$ is a likelihood for a model to have θ as parameters, considering a dataset X of n samples, defined as a joint probability distribution over all data points from X. MLE tries to find the maximum likelihood for X by maximizing $\mathcal{L}(X, \theta)$ using an optimization technique over θ, such as gradient descent.

With an assumption that all data points were obtained independently, the likelihood can be defined as a product of marginal probabilities as follows:

$$\mathcal{L}(X, \theta) = \prod_{n=1}^{n} p(x_i, \theta) \tag{3.4}$$

where x_i is a data point from X and $p(x_i, \theta)$ is a probability of x_i coming from a distribution of a model with parameters θ.

VAE is an explicit method, which means the likelihood function is explicitly available for the training and we could theoretically obtain the maximum likelihood parameters using analytical methods. However, the maximum likelihood function is generally intractable in practice and it has to be approximated. To solve that, VAE uses lower bound estimator of KL divergence [9] $D_{KL}(P_{data} \mid\mid P_{model})$ between the target distribution P_{data} and the model distribution P_{model} (equation 3.5). It is used with a reparameterization trick so that an optimizable approximation of the objective function is yielded and the gradient descent optimization is possible [4]. Since the KL divergence is a measure of how two probability distributions are similar to each other, to maximize the likelihood, it is analogous for the model to minimize $D_{KL}(P_{data} \mid\mid P_{model})$.

KL Divergence between two continuous distributions P and Q is defined as:

$$D_{KL}(P \mid\mid Q) = \int_x P(x) \log \frac{P(x)}{Q(x)} \, dx \tag{3.5}$$

GAN objective can be characterized by a symmetrized mean of two KL Divergences, the JS Divergence [10], which overall yields better results than a single KL Divergence. A JS Divergence $D_{JS}(P \mid\mid Q)$ between two distributions P and Q is defined as:

$$D_{JS}(P \mid\mid Q) = \frac{1}{2} \, D_{KL}(P \mid\mid \frac{P+Q}{2}) + \frac{1}{2} \, D_{KL}(Q \mid\mid \frac{P+Q}{2}) \tag{3.6}$$

where D_{KL} is a KL divergence defined in equation 3.5.

GAN is an implicit method, which means that it samples data from their distribution and that the generator does not need to explicitly know the target distribution, because its parameters are adjusted based only on the discriminator's loss. Since the loss function (equation 3.2) can be thought of as maximizing the conditional probability $P(Y = y \mid x)$, where Y indicates whether x comes from the real data distribution ($y = 1$) or from the generated one ($y = 0$), the likelihood objective is projected into GAN through the discriminator. For any given generator, this discriminator likelihood objective can always reach the optimum when both networks have enough capacity. The generator can then be updated to improve the objective (equation 3.1) for the optimal discriminator and thus make its distribution converge towards the real data distribution [6]. Therefore, contrarily to VAE, there is no intractable integral which needs to be approximated.

For VAE, approximating the likelihood of a dataset generally means a limitation in generating perfect samples. For example, images generated by VAE might not look realistic as they often appear to be blurry. Yet, the objective and the explicit likelihood approach is naturally quite stable. On the other hand, GANs are theoretically capable of producing more realistic data thanks to its objective function. With that being said, they are notoriously very hard to work with and they are very unstable when it comes to their training and many unwanted situations which might occur due to it [11]. Some of the improvements to the vanilla GAN architecture which combat the issues are discussed later (section 3.5).

Another disadvantage of the adversary training is the inability to properly evaluate the trained network or the samples generated by it. Apart from a subjective realness evaluation of every generated sample by a human, GANs do not provide any natural metric which would tell us how good it is. This is due to the fact that the loss function of the generator used for synthetic data sampling only expresses how good the generator is at fooling the discriminator. However, this alone does not provide any information about the quality of the samples alone. Contrarily, the loss function of VAE expresses a reconstruction loss on an input after its encoded into the latent space and decoded back, providing us some information about the model once it is trained. The evaluation used in this thesis is described in detail in chapter 5.

3.5 GAN Improvements

As already mentioned, vanilla GAN algorithm (section 3.3) has proven to be highly unstable. This section first points out the most frequent problems [12] which make a GAN training difficult. Then, it describes improving techniques which aim to solve these problems by architectural changes. The techniques presented are used by various state-of-the-art algorithms (including the CTGAN algorithm, thoroughly described in 4.2), to increase the overall performance and stability.

Although there are also many improvements to vanilla VAE algorithm (section 3.2), only GAN techniques are presented. Techniques which have been proposed for VAE are not used in the implementation part (chapter 6), so they do not need to be discussed any further.

Common Issues

The issue presented the most when it comes to GANs is a modal collapse. A collapse might develop when the model tries to learn a multi-modal data distribution. Since the generator is only backpropagated whether it fooled the discriminator or not, it might learn to ignore most of the data modes and to generate very similar data points which would fool the discriminator. Because of that, the generator only explores a small part of the data space and although it might produce high quality data, the output is very little diverse.

Another common obstacle is a non-convergence of a model. Although GANs are theoretically proven to converge towards the optimal solution, it is immensely difficult to achieve that in practice in a generic manner. The minimax training approach is unreliable and the and all models are extremely dependent on their hyperparameters.

This is closely related to a vanishing gradient problem. To help the training stability, the discriminator is often in advance to the generator. This can lead to the discriminator overfitting, meaning the loss function of the generator does not provide a sufficient gradient for learning, preventing the discriminator from converging.

There are many improvements which aim to solve the listed problems [7], but only few are presented further in this thesis. It would be out of the scope of this thesis to describe all important improvements to the vanilla GAN architecture which have been proposed. As a result of that, the thesis only presents the techniques adopted by the CTGAN algorithm as they are important to justify the choice of a GAN-based algorithm.

WGAN

Many GAN adaptations define new objective function to improve the training stability of the algorithm. Wasserstein GAN (WGAN), proposed in 2017 [8], significantly reduces the chance of a modal collapse and it does not require careful balancing of both adversaries. To measure the divergence of a real and a generated distribution, WGAN [8] uses a Wasserstein distance. It is often referred to as the Earth-Mover distance and it intuitively determines how mass we have to move to transform one distribution to the other. Formally, a Wasserstein distance $D_W(P \mid\mid Q)$ between two distributions P and Q is defined in the original publication as follows:

$$D_W(P \mid\mid Q) = \inf_{\gamma \in \Pi(P,\ Q)} \mathbf{E}_{(x,y)\sim\gamma}[\ ||x - y||\ ] \tag{3.7}$$

where $\Pi(P,\ Q)$ is the set of all joint distributions $\gamma(x, y)$ of marginal distributions P and Q.

Unlike the cross entropy loss and the JS divergence, the Wasserstein distance provides strong gradient when the distributions are very far from each other or do not overlap at all. Additionally, the distance increases and decreases much more smoothly than it does in case of the traditional divergences and it prevents potential instabilities in the gradient descent training process.

The infimum in the equation 3.7 is intractable, so the Wasserstein distance has to be transformed first in order to yield an appropriate loss function. To do that, the

authors of WGAN use a Kantorovich-Rubinstein duality. The transformation itself is complicated and it is therefore simplified for the purpose of this thesis. Considering a real data distribution P_r and a generated data distribution P_θ, the Wasserstein distance between both distributions $D_W(P_r, P_\theta)$ is transformed as [13]:

$$D_W(P_r, P_\theta) = \frac{1}{K} \sup_{||f||_L \leq K} \mathbf{E}_{x \sim P_r}[f(x)] - \mathbf{E}_{x \sim P_\theta}[f(x)] \tag{3.8}$$

where K is a Lipschitz constant. K has to satisfy:

$$||f||_L \leq K \tag{3.9}$$

When the condition 3.9 is satisfied, the function f is K-Lipschitz continuous. The loss function $L_{WGAN}(P_r, P_\theta)$ can now be formulated. As an objective, WGAN tries to find the best f_w from a set of K-Lipschitz continuous functions $\{f_w\}_{w \in W}$. With respect to that, the loss function is formally defined as:

$$L_{WGAN}(P_r, P_\theta) = \max_{w \in W} \mathbf{E}_{x \sim P_r}[f_w(x)] - \mathbf{E}_{z \sim P_r(z)}[f_w(G_\theta(z))] \tag{3.10}$$

where G_θ is a WGAN generator with parameters θ.

The discriminator in WGAN is referred to as a critic (C), because it learns to compute the Wasserstein distance between the distributions instead of discriminating real data samples from fakes. The slightly updated architecture of WGAN with a notation from the equation 3.10 can be seen in figure 3.4.

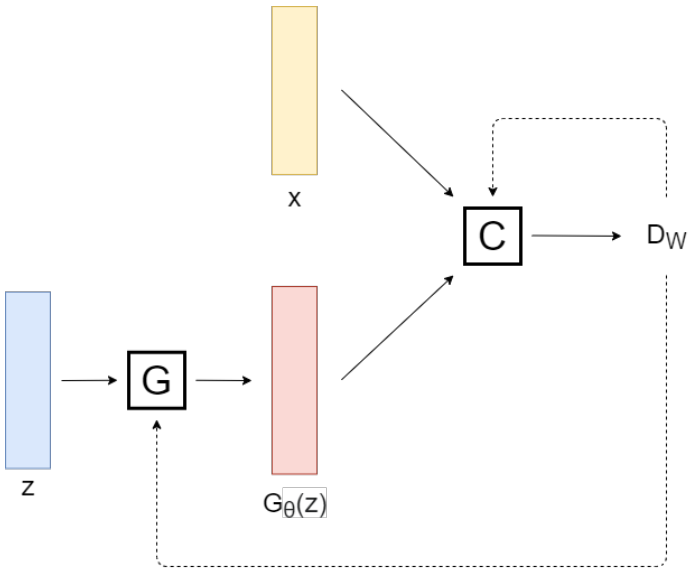


Figure 3.4 Architecture of GAN

The one last problem this approach comes with is how to preserve the K-Lipschitz continuity of f_w during the gradient descent training. To solve that, the WGAN learning algorithm clamps the weights w to a small window (for example, the original implementation uses a rule $|w| \leq 0.01$). This makes the space of all weights w very compact and maintains the K-Lipschitz continuity. [13]

WGAN-GP

The weight clipping trick in WGAN algorithm is not perfect (even according to the authors of WGAN paper) and a generally better solution was proposed in 2017 [14]. WGAN-GP (WGAN with a gradient penalty) does not use a hyperparameter hard-clamping of the weights at all and instead, it introduces new modified loss function for the WGAN critic.

In short, it is necessary for a function f to only have gradients with a norm lesser or equal to 1. To enforce the constraint, a gradient penalty is added to the vanilla WGAN loss function (equation 3.10). The new loss function $L_{WGANGP}(P_r, P_\theta)$ is thus simply defined as:

$$L_{WGANGP}(P_r, P_\theta) = L_{WGAN}(P_r, P_\theta) + GP \tag{3.11}$$

where GP is the gradient penalty.

The gradient penalty corrects the discriminator when its gradient norm moves from 1. Since the loss would be intractable for all x, it is only computed for a gradient norm of randomly chosen samples $\hat{x}$. GP is defined as follows:

$$GP = \lambda \, \mathbf{E}_{\hat{x} \sim P_{\hat{x}}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2] \tag{3.12}$$

where D is the WGAN critic and λ is a model hyperparameter, referred to as gradient penalty coefficient.

Although WGAN-GP adds another computational complexity to the original WGAN loss function, it has proven to perform significantly better than the original algorithm [14] and it is generally a good objective when our goal is to generate high-quality data.

Packing

Packing [15] is another technique which fights the modal collapse issue. It was introduced in 2018 (along with a PacGAN algorithm) and as the paper suggests, it significantly decreases the chance of a modal collapse when it is applied to a vanilla GAN architecture.

The idea of packing is very simple. Instead of using a discriminator $D(X)$ which would take one data point X as an input and classify it (whether it is real or fake), PacGAN uses an augmented version of the same discriminator architecture $D(X_1, X_2, ..., X_m)$ which takes m data points as an input and classifies them all together. To keep the

classification meaningful, all m samples are always drawn from the same distribution, either the real one or the generated one. The generator stays unmodified.

According to the authors, distributions with more modal collapse are penalized by the augmented discriminator. Additionally, it makes a training process smoother, because the output of the discriminator is always based on several different data samples.

Adam Optimizer

Adam [16] is an optimization technique that is not solely related to GANs so the thesis does not describe it thoroughly. Since it is basically an extension to the classical gradient descent optimization, it can be used in any deep learning algorithm, including GANs, to improve the learning efficiency.

There are two main features which help the algorithm to be faster and more efficient, adaptive learning rate and momentum. The first allows Adam to adjust the learning rate as the learning goes, improving its performance on problems with sparse gradients. The second helps to keep a relative direction and thus to be more resistant to noise.

Chapter 4

Tabular Data

Most of the generative deep learning algorithms that have been proposed focus solely on tasks related to images. Deep learning algorithms have no problem handling even large high-dimensional inputs and the uniformity of visual data being represented by pixels makes generative algorithms extremely effective. Contrarily to such data, tabular data are usually represented by low-dimensional inputs, making their distribution theoretically even easier to learn. With that being said, they have many characteristics which cause synthetic tabular data to be generally difficult to generate.

4.1 Tabular data specifics

Each dimension of a tabular data space is defined by a different data column. The column might either be discrete or continuous and each one of them has its own properties. This section presents important issues which a generative model has to take in count in order to appropriately learn a joint distribution of any tabular data.

Data distributions

Single table columns often do not follow any standard easily-defined probabilistic distribution, such as Gaussian. Since table columns can describe any real-world property and each column within a table might even come from a different source, they often vary a lot from each other even within the same dataset and it is greatly difficult to define any generic prior distribution. Additionally, many columns have multiple modes which makes it even more difficult for the generative model to learn the underlying distribution.

Imbalanced columns

As mentioned in [17], majority of the discrete columns are highly imbalanced. The same trend can be observed in the enclosed notebook where the marginal column distributions are plotted. This can cause severe overfitting of the majority class when regular random sampling of the training data is implemented. The data points which

contain a minority class values would be hardly ever selected, causing the model to almost ignore them entirely.

Data types

Table columns can be of a different data types. While some of them, such as integers or floats, are easily transformed to serve as an input for a neural network, the more complicated ones are not. The data thus require preprocessing and there has to be a clear way to transform a table row into a numeric representation, so that a a generative model is able to learn from it. Also, this transformation has to be bidirectional, meaning a synthetic row can be obtained from a model's numeric output.

Constraints

Table columns are often constrained, whether it is only by a given range of values or with respect to another column. A model does not understand what table columns represent in reality and even when it is trained on logically correct data, the correctness of synthetic samples is not guaranteed. Although this does not directly affect the training process, there has to be a modification to the sampling process.

Null Values

Missing values can complicate both the training and the sampling. They have to be imputed for most deep learning architectures, so there is more preprocessing work required before the training. When sampling synthetic data, on the other hand, we often have to create missing values in order to imitate the real dataset in all of its aspects.

4.2 CTGAN

CTGAN (Conditional Tabular GAN) is a state-of-the-art algorithm published in 2019 with a GAN based architecture designed for tabular data generation. The following section covers all components and techniques described in [17] and shows its benefits.

Algorithm Selection

The algorithm selection is not based on any custom evaluation of different approaches. Instead, it is based on the theoretical part of this thesis and consultations with employees of the bank. Section 3.1 shows that both VAE and GAN based algorithms can be extremely effective when it comes to data distribution learning. Although they both have limitations, they can perform extremely well even for difficult tasks and their recent applications prove that deep learning methods are often an excellent choice [3].

Apart from the selected algorithm, GANs and generative deep learning approaches in general have already demonstrated that they can create powerful tools for tabular data generation [18, 19]. CTGAN differs from the older algorithms in several architectural aspects and based on the benchmark published with the algorithm, it outperforms all other tabular data focused generative methods.

Except for this algorithm, [17] also introduces its version of an algorithm derived from VAE (3.2), which is called TVAE (Tabular VAE). Although it performs slightly better than CTGAN on a presented benchmark, the algorithms can be expected to perform similarly overall as there is more flexibility in the CTGAN architecture and the benchmark only uses a single hyperparameter setup. As a result of that, the choice of the architecture derived from GAN (3.3) over the one derived from VAE is based on one of the advantages of the adversarial training. According to the thesis assignment, the selected algorithm should be able not only to augment real dataset, but also to completely anonymize them during the process. This is much easier to achieve with an architecture derived from GAN, because no real data are ever introduced to its generator and synthetic data can thus be considered as differentially private. In contrary, VAE is trained with its real data reconstruction error, so it cannot be strictly defined as a differentially private algorithm.

CTGAN features multiple techniques which aim to eliminate issues related to adversarial training (section 3.5) and thus is a relatively stable and generic algorithm. This makes it easier to successfully train different synthesizers for different production datasets when the dataset is appropriately preprocessed. It also means that no advanced knowledge is required to train a model reasonably well.

Additionally, the algorithm is a part of SDV framework (6.2) and thus is even easier to implement. It provides high-level API for easier implementation and the algorithm used can easily be substituted by a different algorithm when needed (such as TVAE). Also, the evaluation is easier thanks to the framework built-in evaluation module, which is used later in this thesis (5.2).

Architecture

Since CTGAN architecture is derived from GAN, it uses two adversarial neural networks (3.3). Both networks are fully connected and they are composed of 2 hidden layers. The networks are trained against each other using the WGAN-GP objective along with the Adam optimizer. The critic also uses the packing to improve the stability. All of the improving techniques are described in detail in section 3.5.

CTGAN also implements multiple specific custom techniques which help the algorithm to process a tabular input and learn the distribution defined by it. All these techniques are described below.

Mode-specific Normalization

Tables can contain both discrete and continuous columns. CTGAN transforms each of the column types differently, so its original inner representation is maintained.

To represent a discrete column, each of its values is encoded as a one-hot vector. For a column of n distinct values, the one-hot vector contains $(n - 1)$ zeros and a single one, for each distinct value at a different position. Thanks to that, all distinct values from a column can be represented by a different one-hot vector. To set an example, the values from a column with 3 unique values would be represented either by vector $(1,\ 0,\ 0)$, vector $(0,\ 1,\ 0)$ or vector $(0,\ 0,\ 1)$.

In theory, all continuous columns would not have to be transformed at all and they could be directly fed into a discriminator. In practice, GANs often use normalization to adjust the input values so that they come from a predefined range.

For example, to normalize a column C to come from a range of $[-1,\ 1]$, each value x from C is min-max normalized as follows:

$$x' = 2\ (\frac{x - x_{min}}{x_{max} - x_{min}}) - 1 \tag{4.1}$$

where x' is a new normalized form of x, x_{min} is the minimal value from C and x_{max} is the maximal one.

Although the min-max normalization can be effective when all columns come from a single-modal distribution, such as normal, it fails when a column follows a more complicated multi-modal distribution. Values which are close to the same mode are normalized to very similar values and the difference between them is negligible. Because of that, the difference between them might have little impact on the weights of the model, even though it could be significant in reality.

To prevent such scenario, the algorithm uses a mode-specific normalization to transform all continuous columns instead. First, a number of modes of the column distribution is estimated by fitting a Gaussian mixture to it. Intuitively, the distribution is split into multiple normal distributions, so that all the normal distributions combined sum up to the column distribution.

Formally, the probability $P_C(x)$ over the column C is a mixture of K normal distributions $(\mathcal{N}_1, ..., \mathcal{N}_K)$, where $\mathcal{N}_i = \mathcal{N}(\eta_i, \sigma_i^2)$. It is defined as follows:

$$P_C(x) = \sum_{k=1}^{K} \phi_i\ \mathcal{N}_i(x) \tag{4.2}$$

where ϕ_i is a weight of the distribution N_i. The weight is not used in the normalization process at all.

Each normal distribution represents an estimated mode and their mean and variance are used to compute normalized values of the column. Every point is normalized with respect to a distribution which has a highest probability in it. A figure 4.1 (originally from [17]) shows a distribution as a mixture of 3 normal distributions, meaning there are 3 modes estimated in total (η_1, η_2 and η_3). Each mode is represented by a mean of the corresponding normal distribution.

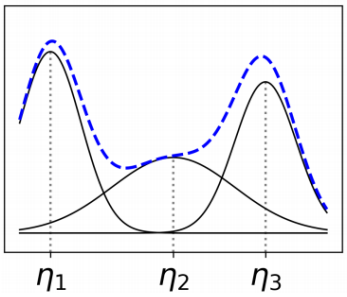


Figure 4.1 Gaussian mixture of a multi-modal distribution

Each value x from a column C is normalized as:

$$x' = \frac{x - \eta_k}{4\sigma_k} \tag{4.3}$$

where η_k is a mode with the highest probability in x and σ_k is a corresponding standard deviation.

Since two values from the same column can now be normalized with respect to a different mode, they can end up having the same norm even when they are originally far from each other. To distinguish a mode which was utilized for the normalization, it is encoded as a one-hot vector described above and the vector is appended to the input.

As a result of the mode-specific normalization, the dimension of the input significantly grows, because each discrete value is encoded as a vector and each continuous column as another vector and a normalized scalar value. With that being said, the tabular input is usually much smaller than a visual one and the transformed representation is mostly composed of one-hot vectors, so it does not cause any computational issues.

A very important characteristics of the whole transformation is its reversibility. Given the vector representation of a row, it is not complicated to obtain the original version which it is supposed to represent. Thanks to that, the generator can output the vector representation so that it can directly serve as an input for the discriminator and when the generator samples new synthetic dataset, its output can be easily translated into the original row format.

Conditional Generating

Conditional generating is not a strictly unique feature to this algorithm. However, it is presented in this section, because it better complements other CTGAN specific features and the described way of conditional generating is only specific to this architecture.

In the vanilla GAN architecture, the generator only requires a randomly generated latent noise as its input (section 3.3). The conditional GANs [20], such as CTGAN, also create a conditional vector for each run of the generator and append it to the latent noise. The condition is therefore projected into the generator input while the randomness is still maintained thanks to the noise.

The conditional vector always represents one unique value out of all discrete columns in the dataset. A column to be represented is selected randomly before each iteration and a value from that column is then selected based on its frequency within that column. Instead of a plain frequency, its natural logarithm is used instead. This means that values which do not occur in the dataset very often are highly prioritized with respect to their absolute frequency, so the frequent values get selected only slightly more often.

The generator then tries to create a data point which contains the value represented by the conditional vector. In order to train him to do that, a cross-entropy loss between a conditional vector and a generated sample has to be added to the original

WGAN-GP loss function (equation 3.11). Apart from the original loss, the generator is also penalized when it generates a different value than the condition, so it gradually learns to project the condition into the generation process.

The cross entropy loss for the condition is defined as:

$$L_{CE} = -\sum_{i=1}^{n} \hat{c}_i \log(\hat{p}_i) \tag{4.4}$$

where $\hat{c}$ is the average over all conditional vectors within the current batch, $\hat{c}_i$ is the value at i^{th} position of $\hat{c}$ and $\hat{p}_i$ is an estimated probability for $\hat{c}_i$.

Each time a new condition is generated, it is also projected into the real data sampling part of the algorithm, so that only samples which do not violate this condition are sampled. Thanks to that, the critic always computes the loss from real and fake samples conditioned by the same value. Simplified CTGAN architecture preview, derived from WGAN architecture (3.4), is presented in figure 4.2.

By this design, the generator is forced to consider all possible values from the discrete columns. If the generator was not conditioned at all, the scarce discrete values would have a negligible influence on the overall distribution learned by the generator.

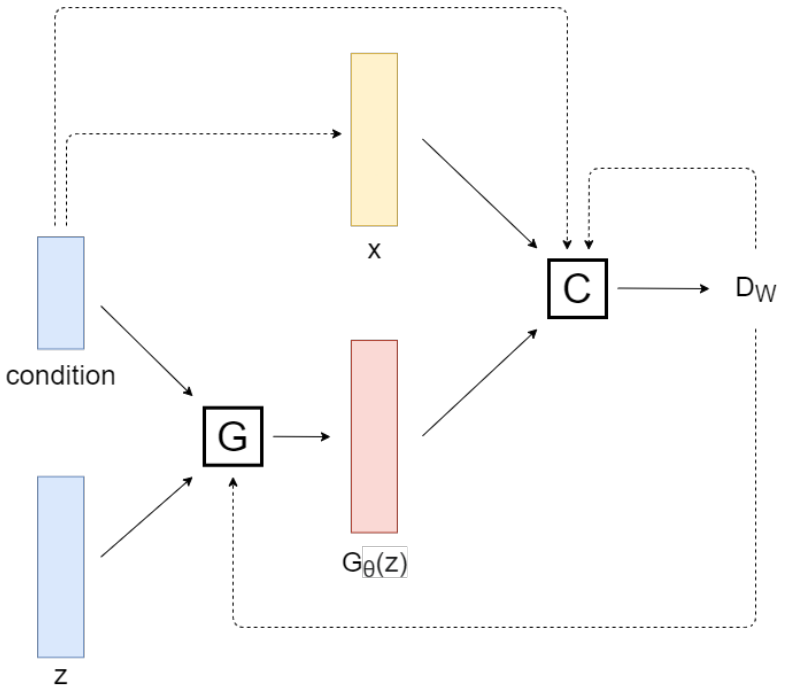


Figure 4.2 CTGAN architecture

Chapter 5

Evaluation Techniques

Although the adversarial training has multiple advantages, it provides no meaningful metric of how well it performs (3.3). Because of that, other metrics have to be defined in order to appropriately evaluate the result.

The metrics described in this chapter are the ones used are the ones used for the final evaluation of the implemented CTGAN model (6). Each metric is theoretically described and a score of the model is provided for each of them.

5.1 Machine Learning Efficacy

ML efficacy is a technique which simulates the ability of a synthetic dataset to be utilized in another task, as a substitution for a real one. To measure a performance of a generative model, completely synthetic dataset is first sampled. It is then used in the task instead of a real dataset and the results are compared to those obtained with the real dataset.

The evaluation process implemented in this thesis is designed to emulate the real use case of the dataset. Since it is originally utilized to predict whether a client will fail to repay a debt or not, a classifier is used as a referential model. The actual model which is adopted by the bank could not be used in this thesis, so a simple decision tree is implemented instead. The decision tree algorithm is not vulnerable to a target column imbalance and it is somewhat universal. Thanks to that, it can serve well as a benchmark even without hyperparameter tuning.

Additionally, features with the highest importance for the utilized decision tree are the same features important to the production model. Therefore, the decision tree should be a relevant benchmark model.

F-Score

F-score has to be defined first so that the evaluation pipeline can be described. It is a widely used performance measure for binary classifiers which is often used to evaluate classifiers which try to predict an imbalanced variable.

It can be computed from a confusion matrix, which is displayed in figure 5.1 (originally from [21]). General F-score F_β is defined a weighted mean of precision and recall with β being a parameter, such that the recall is β times as important as the precision.

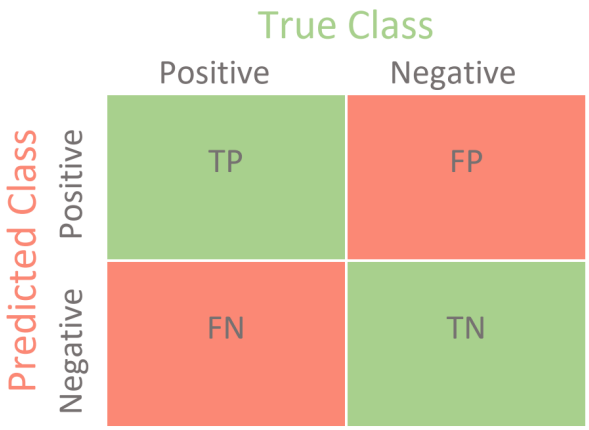


Figure 5.1 Binary classification confusion matrix

Precision is defined as:

$$precision = \frac{TP}{TP + FP} \tag{5.1}$$

and it tells us how many samples classified as positive are in fact positive.

Recall is defined as:

$$recall = \frac{TP}{TP + FN} \tag{5.2}$$

and it tells us how many positive samples were classified as positive.

This thesis only uses a generalized version of F_β with β equal to 1. It means precision and recall are equally important in F_1. The β parameter is not considered any further in the thesis and F-score is meant to be F_1.

As a harmonic mean of precision and recall, F_1 is defined as:

$$F_1 = \frac{2}{precision^{-1} + recall^{-1}} \tag{5.3}$$

Evaluation Process

Details of the evaluation process can be seen in figure 5.2. A real dataset D_{real} is required as an input. D_{real} is randomly split into two subsets, a train set D_{train} and a test set D_{test}. A generator G is then trained using D_{train} and a new completely synthetic dataset D_{synth} is sampled with it. After that, two decision tree classifiers C_{real} and C_{synth} are trained with D_{train} and D_{synth} respectively to predict client failure. Both classifiers are evaluated with D_{test} and an F-score is computed for each of them (F_{real} for a real dataset, F_{synth} for a synthetic dataset). ML efficacy score S is then obtained as follows:

$$S = \frac{F_{synth}}{F_{real}} \tag{5.4}$$

The optimal value for S is 1, meaning both classifiers perform identically. Although scores over 1 mean the synthetic data trained model performs better, it is not the goal of data generation. Instead, the model should create data which are as similar to the original dataset as possible.

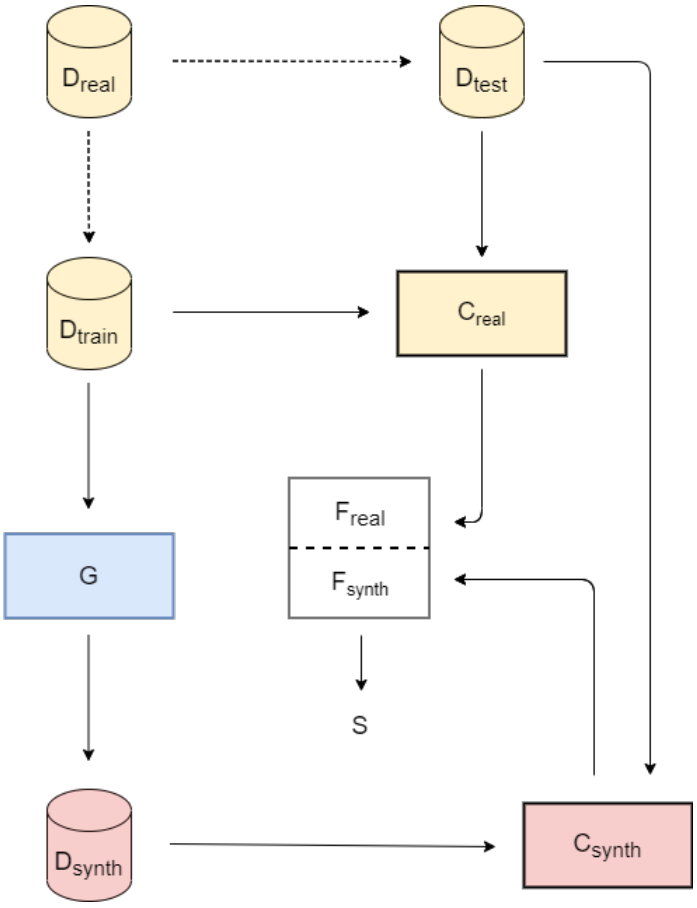


Figure 5.2 ML efficacy evaluation pipeline

5.2 SDV Evaluation

The SDV framework utilized for the implementation (section 6.2) offers multiple evaluation metrics. This section presents all of these metrics used in the enclosed implementation. The metrics can be divided into 3 categories, there are 2 tests in each of the categories.

Statistical Metrics

Each metric runs a statistical tests on both real and synthetic data and compares the results.

- *CS Test* — Uses a chi-squared (CS) test to compare the distributions of all discrete columns. Frequencies from the real dataset contingency table are compared against its synthetic counterparts.
- *KS Test* — Uses a Kolmogorov–Smirnov (KS) test to compare the distributions of all continuous columns. The test works with Cumulative Distribution Functions (CDF) of all columns and the resulting score of the test is an average distance between real and synthetic CDF over all continuous columns.

Detection Metrics

Each metrics trains a ML classifier and then tries to discriminate whether the data belong to the real or the synthetic dataset. Each data point is first assigned a flag that indicates which dataset it originally belongs to. Then, all data points are mixed together, a ML model is trained to predict the flag. The final evaluation is computed as $1-\hat{S}$, where $\hat{S}$ is an average score of multiple models, where each model is validated on a different subset of the mixed dataset.

- *SVC Detection* — Uses a Support Vector Classifier (SVC) for the detection.
- *Logistic Detection* — Uses a logistic regression model for the detection.

Divergence Metrics

Each metric computes a divergence of the synthetic data from the real data. Both metrics use a KL divergence for the computation (equation 3.5).

- *Discrete KL Divergence* — Evaluates all discrete columns.
- *Continuous KL Divergence* — Evaluates all continuous columns.

All metrics output a single scalar between 0 and 1, with 0 being the worst possible score and 1 being the best. An aggregated score is also computed at the end of the evaluation and it is the average of all scores obtained from the separate tests above.

5.3 Visual Evaluation

Both ML efficacy (5.1) and SDV metrics (5.2) output a single score, which indicates the quality of a synthetic dataset. Synthetic data can also be evaluated manually by plotting certain characteristics of synthetic and real data and comparing these characteristics to each other.

For each visual comparison, the enclosed implementation first selects two different subsets of data rows. The subsets can either come from the same dataset, when the goal is to compare distributions conditioned by the target variable, or from different datasets, when the goal is to compare the real dataset to the synthetic one. Then, the marginal distribution of each column is plotted for both subsets against each other.

Chapter 6

Implementation

This section presents details of the enclosed implementation. It serves as a proof of concept for the CTGAN model (thoroughly described in 4.2) selected in the analytical part of this thesis.

6.1 Technologies

Whole implementation uses Python as a programming language. There is a framework (6.2) in Python which offers an easy CTGAN algorithm high-level implementation and apart from that, there are many data-related libraries which make the data processing and the implementation easier. The version of the Python used is 3.7.0.

The enclosed document which contains the implementation is in IPython (.ipynb) format [22]. It is a JSON-based format which combines Python programming language with other programming languages, such as Markdown. This allows interactive computing and visualizations in Python to be in a presentable form, thanks to the Markdown formatted text. In this thesis, markdown is the only other programming language used for the implementation apart from Python.

Another advantage, important for this work, is that the IPython document preserves its Python output when being transferred and it can be exported into several read-only formats, such as HTML. This is very important, because the enclosed code requires the original dataset to work, and thus is not runnable. The train dataset for this thesis is a production dataset provided by the bank, so the data are all highly sensitive and they cannot be exported from the bank perimeter.

6.2 SDV Framework

Synthetic Data Vault (SDV) [23] is a Python framework that enables synthetic data generation. In addition, it implements several other functions, such as data evaluation metrics and model benchmarks, from which this thesis uses the evaluation metrics (section 5.2). The library is open-source and it is free to be used by anyone, even for commercial purposes.

What is the most important, SDV contains a tabular module, which can be used for the tabular data generation. It offers multiple statistical and ML generative algorithms, including the CTGAN. The algorithms have universal high-level API and they can be optimized by defining different hyperparameters. Thanks to that, it is very easy to use a different algorithm if needed in the future, even thought this demonstration only implements CTGAN.

6.3 Procedure

This section describes the demonstrative data anonymization pipeline. Whole process is pictured in figure 6.1.

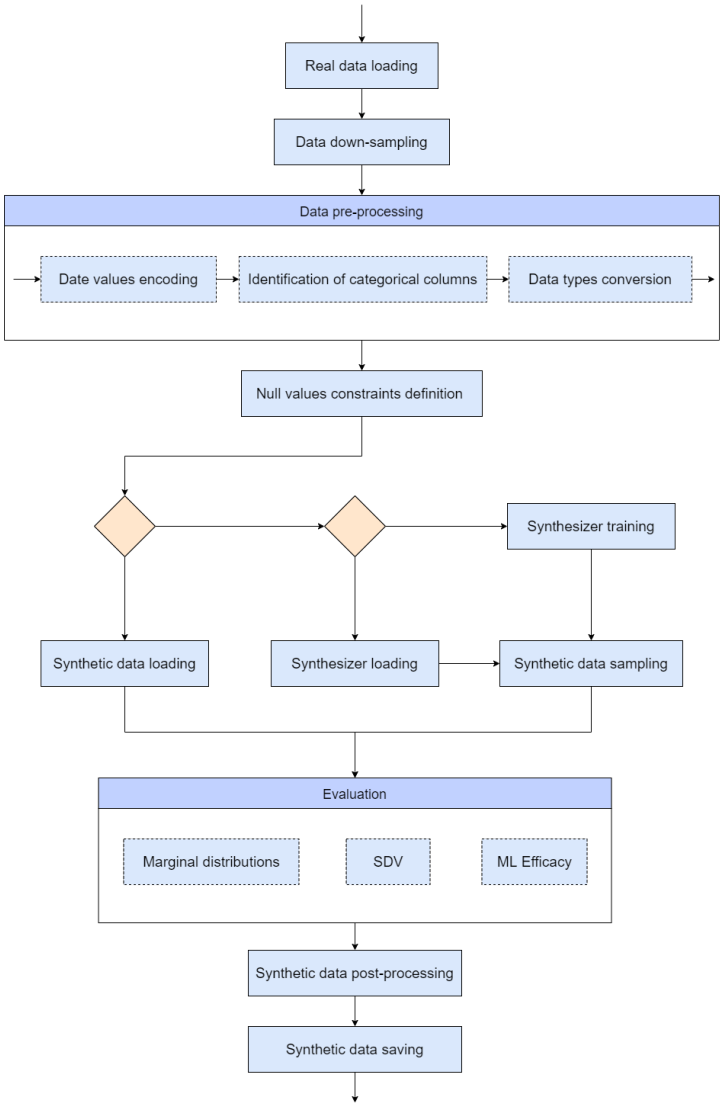


Figure 6.1 Data anonymization pipeline

First, the real dataset is loaded from a CSV file. Although the data are loaded by directly from a database in practice, the demonstrative implementation only works with CSV files. It contains information about clients of the bank and it is originally used as a train dataset for a binary classification model. The target variable is *GOOD_BAD*. Apart from the target variable and *ID* column which is dropped later in the preprocessing phase, the dataset has 24 features.

It is down-sampled and preprocessed, so it can be used for the synthesizer training. Data constraints are defined and a CTGAN model is trained. The constraints defined before the training are projected into the data sampling process by a reject-sampling technique and a synthetic dataset is generated. It is then evaluated against the original dataset and all results are displayed. Finally, all columns of the synthetic dataset which are in a different format due to the preprocessing phase are converted back and the dataset is saved.

Both model training and new data reject-sampling can be skipped. In such case, a pretrained model or an already synthesized dataset respectively are used instead. This option was used not only in the enclosed notebook, but also for debugging.

Sampling

The dataset originally contains 233 035 rows, so it is randomly down-sampled with respect to *GOOD_BAD* column. The dataset has highly imbalanced target variable, with 94 % of the rows being positive and only 6 % of the rows being negative. Although the CTGAN synthesizer can learn from imbalanced columns, all negative samples (minority class) are kept and only the positive ones (majority class) are sampled to 20 %. Thanks to that, not only is the classifier used for the ML Efficacy test trained with more balanced dataset, as it is in the practice, but also the visualization of negative samples are more relevant. Additionally, there are still enough samples left after the process and both training and evaluation algorithms require significantly less time to complete. After this part, the dataset contains 57 810 rows, with 76 % of the rows being positive and 24 % of the rows being negative.

Preprocessing

The preprocessing part is relatively short, because all values in the dataset are generated by machine and the dataset is naturally very clean. The date column converted into a numeric format, so that the generator can better learn its distribution and all categorical columns are identified. This is extremely important, because the CTGAN algorithm does not process categorical the same as continuous ones. After categorical columns are identified, all columns are converted to correct data types.

The algorithm only distinguishes between discrete and continuous columns, but there is no natural way of identifying a column as categorical or ordinal. This is very easy to accomplish in case of the presented dataset, because all columns marked as discrete only contain two values and thus probably represent binary categories. However, this issue is open to any solution which works for each specific dataset. Generally speaking, the best approach would probably be to treat ordinal columns as continuous, but it definitely depends on other column characteristics, such as number of unique values.

Training and Reject-sampling

Next part of the notebook analyses all null values in the dataset. This method is slightly more complicated as the anonymization process has to preserve all relations between null values in different columns. Although CTGAN algorithm is capable of working with null values, the goal is not to create a dataset with imputed null values. Synthetic data should rather be as similar to real data as possible, including the null values appearance. The training and sampling process is pictured in figure 6.2

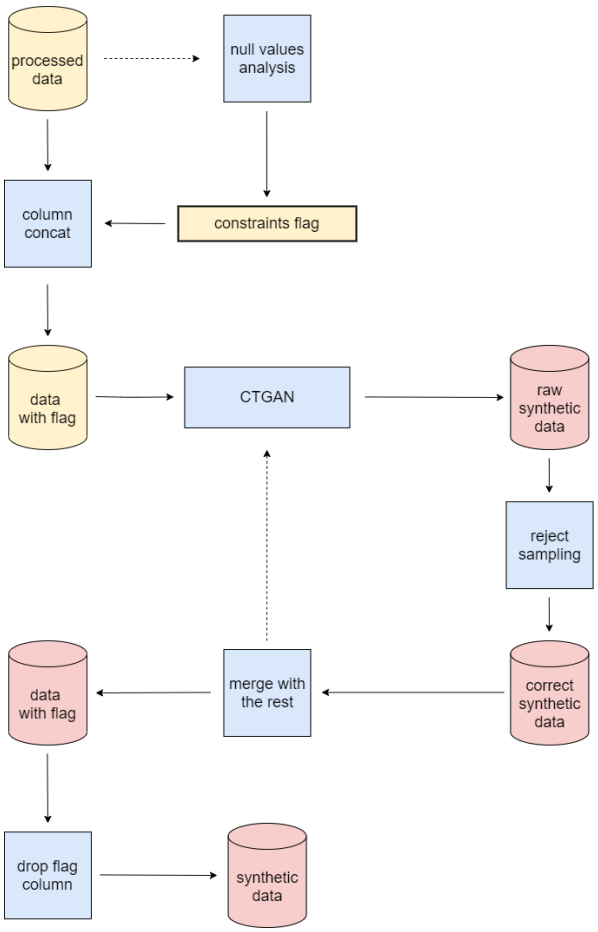


Figure 6.2 Training and reject-sampling process

The presented constraints mechanism is based on a number of null values in each column. Considering the number of rows in the dataset, it is very likely that multiple columns with equal amount of null values only appear together. Therefore, all columns are compared to each other with a consideration of their null values appearance, so that null values constraints (NVC) can be defined for each of the columns. For each column, NVC define which other columns have to always be null and which of them have to always be filled in case that the column is null itself.

Based on the analysis, a new column *NULL_FLAG* is defined to indicate which NVC a row maintains and the column is added to the dataset. The extended dataset is then used to train the CTGAN model, so it learns the distribution with respect to *NULL_FLAG*. Then, the model keeps sampling small synthetic datasets. These datasets get reject-sampled, meaning that all corresponding NVC are applied to each row based on its *NULL_FLAG* value and all the rows which violate any of NVC are dropped. The new dataset generation and the reject-sampling continues until a predefined number of synthetic rows is generated. Finally, *NULL_FLAG* column is dropped from the dataset as it is not needed any further.

Although the reject-sampling can theoretically take extremely long time to complete when there are a lot of constraints that have to be respected, the sampling is very fast in the demonstration as it only rejects minority of rows. This is probably thanks to the fact that the model learns the data distribution well, so it samples new rows naturally well with respect to the constraints.

It should be noted that the model trained for the sampling has default hyperparameters and no tuning technique is implemented. This causes no major issue as the main purpose of the implementation is being a proof of concept.

Data Ratios

There are many different subsets of both datasets used during the procedure. For better clarity, figure 6.3 and table 6.1 display all data subsets used.

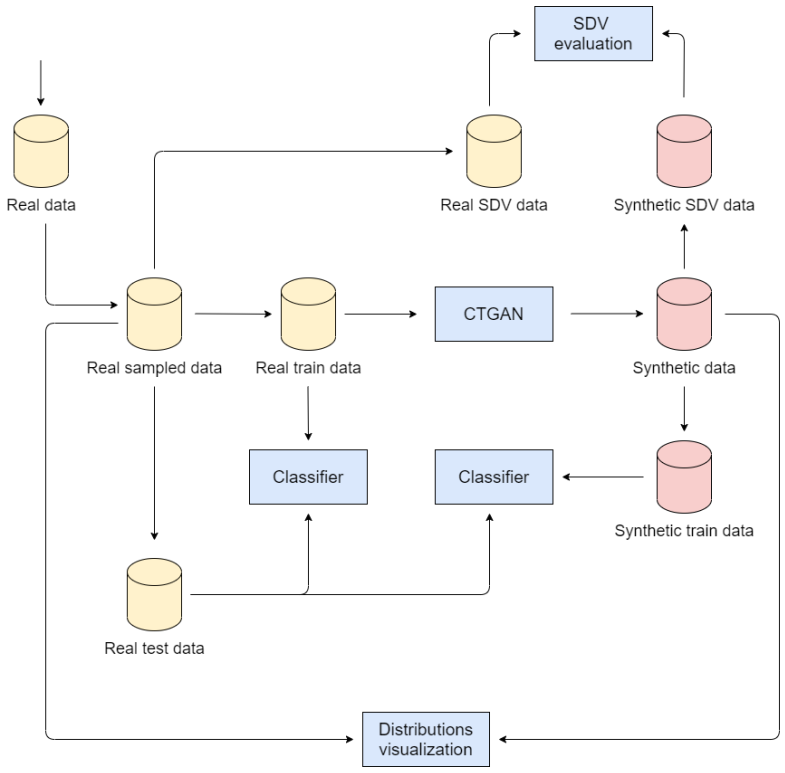


Figure 6.3 ML efficacy evaluation pipeline

Dataset	Number of rows	Relative ratio
Real data	233 035	
Real sampled data	57 810	1.0
Synthetic data	57 810	1.0
Real train data	40 467	0.7
Synthetic train data	40 467	0.7
Real SDV data	28 905	0.5
Synthetic SDV data	28 905	0.5
Real test data	17 343	0.3

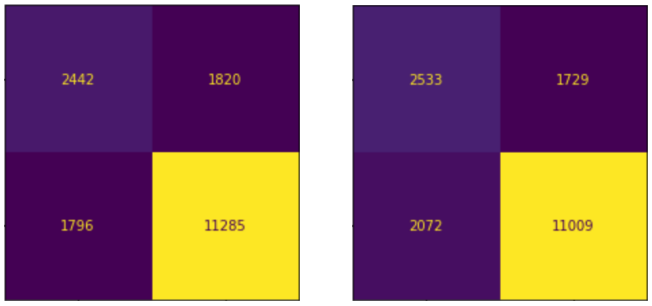
Table 6.1 Sizes of different data subsets

6.4 Results

Generated dataset is evaluated with all techniques presented in chapter 5. There are 57 810 synthetic rows sampled for the evaluation, because all the evaluation techniques should work with two equally large datasets. The only exception is the ML Efficacy (5.1), which utilizes synthetic data only for a classifier training. For its purpose, the synthetic dataset is randomly down-sampled to 40 467 rows in order to have the same size as the real data train set.

ML Efficacy

The results are obtained by the procedure described in 5.1. The ML efficacy score is 0.9943. Figures 6.4 displays confusion matrices (5.1) of both real and synthetic data.



(a) Real **(b)** Synthetic

Figure 6.4 Confusion matrices of both datasets

Although the confusion matrices are slightly different, the score alone indicates excellent results, showing that a classifier trained with synthetic data has almost identical performance as the one trained with real data.

SDV Evaluation

The table bellow presents the results obtained from all SDV metrics defined in 5.2. The average score aggregated from all metrics is 0.6595.

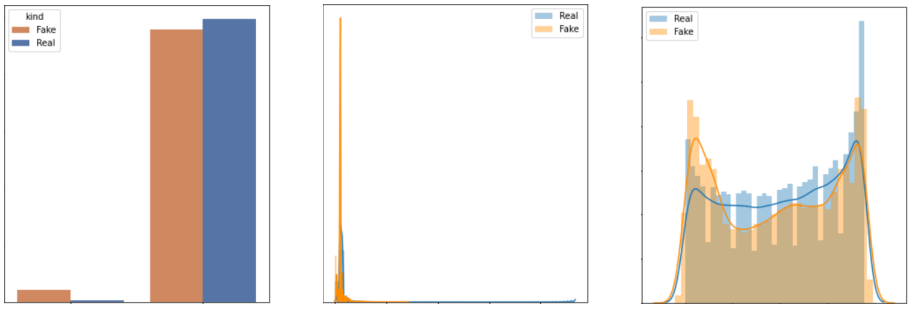
CS Test	0.4121
KS Test	0.8868
SVC Detection	0.7617
Logistic Detection	0.6668
Discrete KL Divergence	0.9155
Continuous KL Divergence	0.3140

Table 6.2 SDV metrics results

The scores of *KS Test* and *Continuous KL Divergence* show that the model learned the distribution fairly well in case of continuous columns. On the other hand, both test which focus strictly on discrete columns, *CS Test* and *Discrete KL Divergence*, show that discrete columns differ considerably. Since all discrete columns are binary, it simply indicates a change in label frequencies across the synthetic dataset. The detection test show good results, meaning it is generally difficult to predict which dataset a sample comes from.

Marginal distributions

Real and synthetic data are plotted against each other so they can be visually compared. Marginal distributions of all columns were compared separately, as described in 5.3. Apart from the enclosed implementation, histograms of all columns across both datasets can be seen in appendix A.



(a) Discrete column **(b)** Flat column **(c)** Multi-modal column

Figure 6.5 Learned marginal distributions examples

Some of marginal distributions are very similar, meaning that the model converged nicely with respect to them and was able to learn the real data distribution reasonably well. However, several columns have extremely flat distributions with one very frequent mode. In their cases, the model generally failed to cover all possible values and only approximated the distribution while keeping even lower variance.

It should also be noted that the model had no trouble to learn marginal distributions which have multiple modes, thanks to the mode-specific normalization described in section 4.2.

Evaluation Summary

Generally, the model achieved significantly better results in case of continuous columns, with an exception of columns with extremely flat distribution. This trend can be seen in both SDV evaluation part and visualizations part above. The model preferred minority class over the majority class, with respect to their original frequency. This was probably caused by the log-frequency sampling implemented by the algorithm (4.2). Although it helps to learn the overall distribution even when the classes are highly imbalanced, the sampling frequency of each class is apparently projected into synthetic data. With that being said, since the ML efficacy test shows excellent results, it does not necessarily have to cause any trouble. The log-frequency sampling can be set off as one of model's hyperparameters and its choice depends on intended use case of the synthetic dataset.

Although some of the marginal distributions were learned incorrectly, it had no significant impact on a classifier in the ML efficacy test. This probably means that omitted values were not originally important for a prediction and their absence thus did not change how a classifier is trained. With that being said, the values should occur in synthetic data in many other cases and the visualization thus shows a potential defect of the model.

Chapter 7

Conclusion

The thesis outlines generative machine learning methods with a focus on synthetic data generation. This chapter summarizes the main contribution with a consideration of the assignment. Then, future improvements to the implementation are discussed as it is the last part of the assignment.

7.1 Contribution

The thesis discusses the idea of using generative methods for data anonymization and augmentation. The generative methods are surveyed with an extensive focus on deep learning methods as they are the best option for such task at the moment. Two main deep learning approaches, VAE and GAN, are thoroughly described and objective is analyzed, considering both their advantages and disadvantages. The theoretical part of this thesis can therefore be utilized by other programmers from the bank or by anyone who would start working in the synthetic data generation field with little prior knowledge. Although the methods described in the thesis are simplified with respect to the original papers, the thesis should provide solid theoretical basis.

An algorithm suitable for the task is selected and it is then trained on a production dataset. The implementation part, which includes simple data preprocessing and model training, serves as a proof of concept for any tabular data anonymization. The trained model can be easily exported and thus serve as synthetic data oracle in the future. Therefore, it can be later used for both data anonymization and data augmentation, as described in the assignment of the thesis. The evaluation part shows that the idea of using completely synthetic datasets instead of real ones is very promising.

The thesis also shows an example of applying more complex constraints to the sampling process (6.3). The mechanism is effective and it handles all constraints correctly. Although there are many use cases when restricting null values is not necessary, the implemented mechanism can also generally serve as a proof that the reject-sampling process does not have to be immensely time consuming when a generator learns the real data distribution well enough.

Thanks to the selected method, the implementation is simple and the training process is not difficult to control, as it might be in case of other deep learning methods. The algorithm is a part of SDV library, which contains other synthetic data related features and opens up several easy improvements of the implementation. These improvements are discussed in section 7.2.

7.2 Future Improvements

This section discusses possible improvements to the implementation which is discussed in section 6.3.

Hyperparameters

Firstly, there is no hyperparameter tuning process in the procedure and it should definitely be added in order to get the best results possible. It is very unlikely for the selected algorithm to have a universal hyperparameter set up with the best performance and the hyperparameter tuning is one of the key techniques which could improve a model by a lot. Although tuning techniques often require immense computational power, it is not expected to be an issue in the future.

On the other hand, hyperparameter tuning is generally very problematic in case of GAN derived algorithms and it cannot be done automatically with a simple evaluation metric. The best technique is an open topic and although WGAN objective (3.5) suppresses many GAN training issues, all technique still requires expertise.

Evaluation

There are multiple metrics used to score the final model, all described in 5. With that being said, each metric only measures certain aspect of the model and even together, they still cannot provide a completely objective rating of the model. Although another technique would not solve the issue, there are many techniques which could be added to provide more complex evaluation.

An excellent example of another metric is an improvement to the ML efficacy technique (described in section 5.1). Even though the technique is relatively complex and it imitates the intended use case of synthetic data, it can still be somewhat narrow. To set an example, the ML efficacy score obtained in the implementation is extremely good (6.4), while there is a slight difference in the confusion matrices obtained by the classifier (6.4). This could mean that the synthetic data classifier prioritizes different features than the real data one, but both options are equally important. To measure this, we could either compute a closeness of the confusion matrices or a closeness of feature importance vector, in case it is important in our situation.

Relational Data Anonymization

The implementation only shows anonymization process for a single table. Since data needed for a certain task are often stored in multiple tables, another step could be an anonymization of a relational structure of multiple tables. This could either be done by implementing SDV hierarchical modeling algorithm (HMA) or by implementing custom recursive algorithm with use of a tabular generative model, such as CTGAN.

Simple Rules Constraints

Lastly, SDV constraints mechanism could be used to maintain trivial relations between columns that have to be respected. Although there is custom constraints implementation for null values in this thesis (6.3), SDV offers faster way to define simple constraints, such as value ranges for certain columns. Although it is not used for the null values in the enclosed implementation for its inflexibility, it works perfectly fine with simple conditions, such as restriction of a time period. The values are not constrained in the demonstration as it is not necessary for its purpose, but they could definitely be useful in a real case scenario.

Bibliography

1. SHARMA, Himanshu. *Generating Data By Drawing*. 2021. Available also from: https://towardsdatascience.com/generating-data-by-drawing-c169fe10632.
2. CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*. 1989, vol. 2, no. 4, pp. 303–314. ISSN 1435-568X. Available from DOI: 10.1007/BF02551274.
3. KARRAS, Tero; LAINE, Samuli; AILA, Timo. A Style-Based Generator Architecture for Generative Adversarial Networks. *CoRR*. 2018, vol. abs/1812.04948. Available from arXiv: 1812.04948.
4. KINGMA, Diederik P; WELLING, Max. *Auto-Encoding Variational Bayes*. 2013. Available from arXiv: 1312.6114 [stat.ML].
5. ROCCA, Joseph. *Understanding Variational Autoencoders*. 2019. Available also from: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73.
6. 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].
7. SALIMANS, Tim; GOODFELLOW, Ian; ZAREMBA, Wojciech; CHEUNG, V.; RADFORD, Alec; CHEN, Xi. *Improved Techniques for Training GANs*. 2016. Available from arXiv: 1606.03498 [cs.LG].
8. ARJOVSKY, Martin; CHINTALA, Soumith; BOTTOU, L. *Wasserstein GAN*. 2017. Available from arXiv: 1701.07875 [stat.ML].
9. KULLBACK, S.; LEIBLER, R. A. On Information and Sufficiency. *The Annals of Mathematical Statistics*. 1951, vol. 22, no. 1, pp. 79–86. ISSN 00034851. Available also from: http://www.jstor.org/stable/2236703.
10. LIN, J. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information Theory*. 1991, vol. 37, no. 1, pp. 145–151. Available from DOI: 10.1109/18.61115.
11. ARJOVSKY, Martin; BOTTOU, Léon. *Towards Principled Methods for Training Generative Adversarial Networks*. 2017. Available from arXiv: 1701.04862 [stat.ML].
12. BARNETT, Samuel A. *Convergence Problems with Generative Adversarial Networks (GANs)*. 2018. Available from arXiv: 1806.11382 [cs.LG].
13. WENG, Lilian. *From GAN to WGAN*. 2019. Available from arXiv: 1904.08994 [cs.LG].

14. GULRAJANI, Ishaan; AHMED, Faruk; ARJOVSKY, Martin; DUMOULIN, Vincent; COURVILLE, Aaron. *Improved Training of Wasserstein GANs*. 2017. Available from arXiv: 1704.00028 [cs.LG].
15. 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].
16. KINGMA, Diederik P.; BA, Jimmy. *Adam: A Method for Stochastic Optimization*. 2017. Available from arXiv: 1412.6980 [cs.LG].
17. SKOULARIDOU, Maria; CUESTA-INFANTE, Alfredo; VEERAMACHANENI, Kalyan; XU, Lei. *Modeling Tabular data using Conditional GAN*. 2019. Available from arXiv: 1907.00503 [cs.LG].
18. 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].
19. LONG, Yunhui; LIN, Suxin; YANG, Zhuolin; GUNTER, Carl A.; LI, Bo. *Scalable Differentially Private Generative Student Model via PATE*. 2019. Available from arXiv: 1906.09338 [cs.LG].
20. MIRZA, Mehdi; OSINDERO, Simon. *Conditional Generative Adversarial Nets*. 2014. Available from arXiv: 1411.1784 [cs.LG].
21. MOHAJON, Joydwip. *Confusion Matrix for Your Multi-Class Machine Learning Model*. 2020. Available also from: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826.
22. PÉREZ, Fernando; GRANGER, Brian E. IPython: a System for Interactive Scientific Computing. *Computing in Science and Engineering*. 2007, vol. 9, no. 3, pp. 21–29. ISSN 1521-9615. Available from DOI: 10.1109/MCSE.2007.53.
23. *Synthetic Data Vault*. [software]. MIT Data to AI Lab., 2020. Available also from: https://sdv.dev/.

Appendix A

Appendix

Figure A.1 Marginal distributions of real and fake data columns

Appendix B

Content of enclosed medium

- `impl` ..... enclosed implementation
  - `thesis.html` ..... read only notebook exported to HTML
  - `thesis.ipynb` ..... source IPython notebook with the implementation
- `text`
  - `thesis.pdf` ..... thesis text in PDF